AN: PAT 2002-242950

TI: Automation data distribution system uses coded datagram for communication between data processing station and automation component

PN: **DE10112843**-A1

PD: 27.09.2001

AB: NOVELTY - The automation data distribution system has a data processing station which is provided with a network address and an automation component (AK) with a network address, which is functionally coupled to the data processing station, with communication between them via a datagram which is fully coded in clear text. DETAILED DESCRIPTION - Also included are INDEPENDENT CLAIMS for the following; (a) a datagram for use in a system for automation data distribution; (b) an automation data distribution method; USE - The automation data distribution system is used for fabrication automation. ADVANTAGE - The automation data distribution does not require the resources of a windows personal computer. DESCRIPTION OF DRAWING(S) - The figure shows an abstract block diagram of an automation data distribution system.

PA: (LANG-) LANGNER COMMUNICATIONS AG;

IN: LANGNER R;

FA: **DE10112843**-A1 27.09.2001;

CO: DE;

IC: G06F-017/60;

MC: T01-D02; T01-J11A; T01-N01A2E; T01-N01D; T01-N02A3B; U21-A05A; U21-C02;

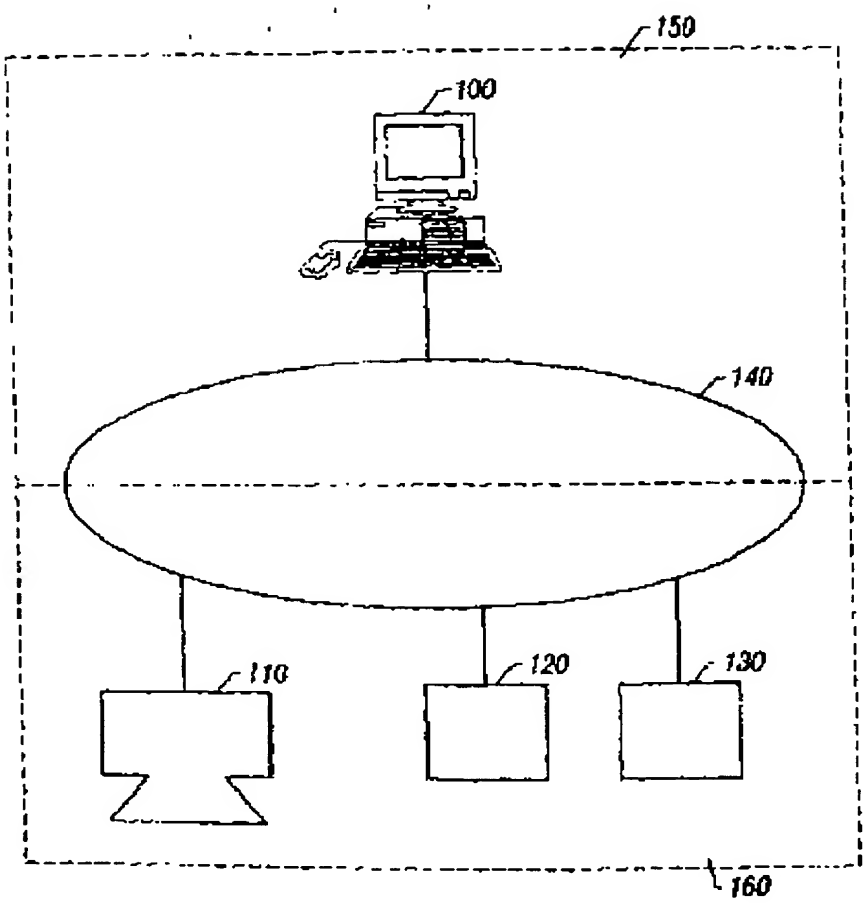
DC: T01; U21;

FN: 2002242950.gif

PR: US0527773 17.03.2000;

FP: 27.09.2001

UP: 10.05.2002


150
100
140
110
120
130
160

02P00693

(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 101 12 843 A 1**

(51) Int. Cl.[7]: G 06 F 17/60

(21) Aktenzeichen: 101 12 843.6
(22) Anmeldetag: 16. 3. 2001
(43) Offenlegungstag: 27. 9. 2001

DE 101 12 843 A 1

(30) Unionspriorität:
527773 17. 03. 2000 US

(71) Anmelder:
Langner Communications AG, 22359 Hamburg, DE

(74) Vertreter:
Betten & Resch, 80333 München

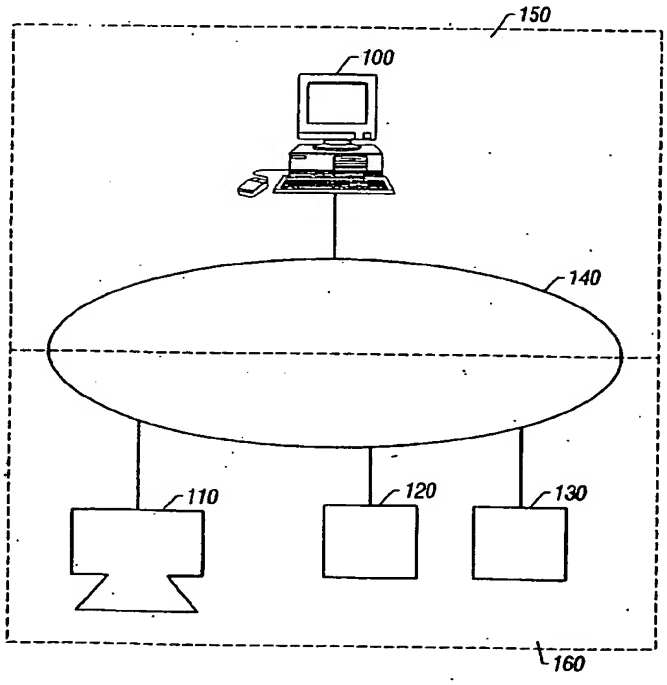
(72) Erfinder:
Langner, Ralph, 22393 Hamburg, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gem. § 44 PatG ist gestellt

(54) System und Verfahren zur Verteilung von Automatisierungsdaten (Automation Data Distrubution, abgekürzt: ADD)

(57) Die vorliegende Erfindung zielt auf ein System und Verfahren zur automatisierten Datenverteilung. Im besonderen bietet die vorliegende Erfindung ein System zur Verteilung von Automatisierungsdaten, welches Systemkomponenten auf der Planungsebene (zum Beispiel kaufmännischen Softwareanwendungen) und Systemkomponenten auf der Steuerungsebene (zum Beispiel Anwendungen der Fabrikautomation) ermöglicht, durch die Verwendung vollständig codierter Datagramme einfach miteinander zu kommunizieren.



DE 101 12 843 A 1

Beschreibung

HINTERGRUND DER ERFINDUNG

1. Umfeld der Erfindung

Die vorliegende Erfindung bewegt sich allgemein im Umfeld der Fabrikautomation. Genauer gesagt beschreibt sie ein Verfahren zur Datenverteilung zwischen Anwendungen der Fabrikautomation und Softwareanwendungen aus dem IT-Bereich.

2. Stand der Technik im Umfeld der Erfindung

Automatisierungstechnik und betriebswirtschaftliche Datenverarbeitung haben sich über Jahre hinweg parallel und relativ unabhängig voneinander entwickelt. Seit geraumer Zeit wird jedoch die Wichtigkeit eines transparenten und zeitnahen Datenflusses zwischen allen Aspekten und Teilbereichen eines Unternehmens erkannt. Fabrikautomation und kaufmännische Softwareanwendungen werden dabei als integrale Bestandteile eines übergeordneten Datenfluss-Prozesses betrachtet. Der Begriff "Manufacturing Execution System" ("MES") – im Deutschen auch der Begriff "Vertikale Integration" – wird benutzt, um diese Integration anzusprechen.

MES ermöglicht es kaufmännischen Anwendungen, direkt auf Produktionsdaten zuzugreifen. Dies ist sinnvoll zur Überwachung der Prozesssteuerung: zum Beispiel für Visualisierung (HMI), Alarmierung, Fernwartung und Fernkonfiguration, und für Leitsysteme ("SCADA"). Desweiteren eignet sich MES für die Sammlung und Auswertung von Prozessdaten für Planungszwecke, zum Beispiel im Rahmen von Produktionsplanungs- und -steuerungssystemen (PPS/ERP) Manufacturing-Mangement-Information-Systeme (MMI) und zur Erfassung von Produktionsdaten. Schliesslich gibt es eine wachsende Zahl von Softwareanwendungen, die direkt mit Automatisierungskomponenten kommunizieren, um Daten von Sensoren und anderen externen Datenquellen zu erfassen. Beispiele umfassen die Gebäudeautomation, Umwelttechnik, und Laboranwendungen, in denen keine Echtzeiteigenschaften gefordert sind.

Zusammengefasst: MES zielt auf einen direkteren, zeitnahen Datenfluss zwischen Automatisierungskomponenten und IT-Softwareanwendungen. In vielen Unternehmen des produzierenden Gewerbes können hierdurch Qualitätsverbesserungen erzielt und Pufferzeiten verkürzt werden. Im gegenwärtigen Wettbewerbsumfeld, das durch Globalisierung, Buit-to-Order (Anfertigung nach Kundenspezifikation) und Just-in-Time-Produktion (Minimierung von Lagerzeiten) gekennzeichnet ist, lassen sich hierdurch wichtige Produktivitätsgewinne erzielen.

Ein problemloser direkter Datenaustausch zwischen der Steuerungsebene (zum Beispiel automatisierungstechnischen Anwendungen) und der Planungsebene (zum Beispiel betriebswirtschaftliche Anwendungen) war bisher praktisch nahezu unmöglich, da beide Bereiche isoliert voneinander und für auf unterschiedliche Zwecke entwickelt wurden. Sie verwendeten für gewöhnlich zueinander inkompatible Verfahren und Paradigmen zur Datenübertragung. So hatten zum Beispiel Übertragungsprotokolle und -medien der Steuerungsebene kein Pendant auf der Planungsebene. Des weiteren wurden auf der Steuerungsebene zahlreiche untereinander inkompatible Übertragungsprotokolle und -medien verwendet, was einem durchgängigen Datenfluss zwangsläufig entgegensteht. Die traditionelle Industrielle Kommunikation wird am besten durch eine Vielzahl hersteller- und gerätespezifischer Übertragungs- und Anwendungsprotokolle charakterisiert (zum Beispiel 3946R, RK512, Modbus, IEC870, Mewtocol, AK, MPI, LSV2, Interbus, Devicenet, MSV2), von denen die meisten in der Office-Welt praktisch unbekannt sind.

Der Einsatz von Industrieprotokollen für einen durchgängigen Datenfluss zur Planungsebene scheitert oft bereits daran, dass eine Überwachung der unteren Protokollschichten erforderlich ist, die jedoch von modernen Rechnerarchitekturen nicht unterstützt wird. So erfordert beispielsweise das M-Bus-Protokoll (ein Protokoll aus dem Bereich der Fernwirktechnik) nach dem Empfang eines Telegramms eine Pause für die Dauer von 11 Bit, bevor ein Antworttelegramm gesendet werden kann. Das ist mit aktiven (gepufferten) seriellen Schnittstellenkarten, wie sie in heutigen Computern vielfach eingesetzt werden, nicht zu machen.

Die Ablaufverfahren (bzw. Paradigmen) von Steuerungsebene und Planungsebene unterscheiden sich grundlegend. Auf der Steuerungsebene werden sequenzielle Abläufe mit Abfragearchitekturen (Master/Slave) verwendet, um ein deterministisches Zeitverhalten und damit Echtzeitfähigkeit zu erreichen. Im Gegensatz dazu benutzen Softwareanwendungen aus dem IT-Bereich oft ereignisgesteuerte Architekturen. Die zyklischen, sequenziellen Programmstrkturen, die auf der Steuerungsebene verwendet werden, können nicht einfach auf die im IT-Bereich üblichen Multitasking-Rechner übertragen werden, da sie einen Grossteil der Rechenleistung beanspruchen würden. Dementsprechend ist es weder sinnvoll noch einfach für einen Windows-PC, mehrere hundert Mal pro Sekunde Sensordaten abzufragen, was erforderlich wäre, um einen Sensor zu überwachen. PCs und PC-Software eignen sich wesentlich besser zur ereignisgesteuerten Verarbeitung von Änderungsdaten anstatt zur kontinuierlichen zyklischen Überwachung von Datenquellen (welches eher die Domäne von speicherprogrammierbaren Steuerungen ist).

Die meisten Industrieprotokolle wurden für einen eng umgrenzten Verwendungszweck entwickelt, d. h. für die Kommunikation mit einem spezifischen Typ von Peripheriegerät, nicht jedoch für den standardisierten Datenaustausch zwischen Systemen unterschiedlicher Architektur. Auch vermischen Industrieprotokolle zuweilen mehrere Schichten des ISO/OSI-Schichtenmodells. Dies stellt ein grosses Hindernis für die Nutzung solcher Protokolle in Office-Anwendungen und modernen Netzwerkumgebungen dar. Ein Protokoll, welches sich über mehrere Schichten des ISO/OSI-Modells erstreckt, ist unflexibel und oft nur schwierig mit anderen Anwendungen und/oder Übertragungsmedien zu verknüpfen. Die Verknüpfung mit Office-Anwendungen wäre zum Beispiel wesentlich einfacher, wenn es "nur" um die Umsetzung von unterschiedlichen Datenformaten auf den höheren Protokollschichten (ISO/OSI Schicht 6 und 7) ginge und nicht gleichzeitig auch um diffiziles Zeitverhalten auf den unteren Protokollschichten (ISO/OSI-Schichten 4 und darunter). Dort jedoch, wo sich Industrieprotokolle über mehrere Protokollschichten erstrecken, resultiert oft eine "Alles-oder-nichts"-Situation, in der die Office-Anwendung Daten mit dem industriellen Peripheriegerät nur über ein komplexes

Gateway austauschen kann.

Ein Ansatz zur Lösung dieses Problems besteht in der Verwendung von definierten Softwareschnittstellen (APIs, = Application Program Interface, Softwareschnittstelle zwischen Anwendungsprogramm und systemnahen Softwaremodulen). Hierein fallen zahkeiche Softwareprodukte zum Zugriff auf Peripheriegeräte über nicht standardisierte APIs, DDE-Server und OPC-Server. Dem Problem inkompatibler Kommunikationsprotokolle wird hierbei durch Abstraktion von den konkreten Datenübertragungsverfahren auf eine höhere Ebene – eine definierte Softwareschnittstelle zum Anwendungsprogramm – begegnet. Deshalb braucht eine Softwareanwendung nur Funktionen aus dem API aufrufen, um mit unterschiedlichen Peripheriegeräten Daten auszutauschen, ohne sich um die Details der Datenübertragung zu kümmern. Ein API-basierender Ansatz – insbesondere, wenn er netzwerkfähig ist – ist an eine bestimmte Komponenten- bzw. Objekttechnologie wie OLE oder ActiveX gebunden. Er ist deshalb auch an bestimmte Betriebssysteme gebunden, die die betreffende Komponententechnologie unterstützen. Eine weitere Einschränkung dieses Ansatzes ist die implizierte Client/Server-Architektur von OPC und DDE, die davon ausgeht, dass ein zentraler Softwareprozess (der Server) mit unterschiedlichen Peripheriegeräten per Feldbus oder seriellen Punkt-zu-Punkt-Verbindungen verbunden ist.

Ein für OPC spezifisches Problem ist seine Komplexität, sowohl im Hinblick auf die Einarbeitung als auch auf die Implementierung. OPC und OLE sind nur mit erheblichem Ressourceneinsatz im Feldgerät zu implementieren. Die Implementierung eines OPC Servers in einem embedded System mit marktgängigen Mikrocontrollern ist schwierig, wenn nicht unmöglich.

Ein anderer Ansatz basiert auf dem Grundgedanken, mit Ethernet und TCP/IP ein durchgängiges Transportmedium für die Datenübertragung zu verwenden. Hier finden sich die unterschiedlichen Bestrebungen, die Ethernet als "Feldbus-Ersatz" propagieren und TCP als das künftige universelle Transportprotokoll für den Zugriff auf Automatisierungskomponenten. Ein grundlegendes Problem dieses Ansatzes ist, dass TCP lediglich ein Transportprotokoll (jedoch kein Anwendungsprotokoll) ist und aufgrund seines hohen Protokoll-Overheads nicht einmal besonders gut für automatisierungstechnische Zwecke geeignet ist (geringer Datendurchsatz und langsamer Verbindungsaufbau/-abbau). Ausserdem ist TCP nicht paketorientiert, sondern datenstromorientiert. Bei datenstromorientierten (auch: zeichenorientierten) Protokollen kann eine einzelne Datenstruktur in mehrere Pakete unterschiedlicher Länge aufgeteilt werden, wobei dann der Empfänger der Daten das Problem hat, Anfang und Ende der Datenstruktur zu erkennen. Hierzu ist mindestens ein weiteres höherrangiges, paketorientiertes Protokoll erforderlich, für das sich jedoch weder in der Automatisierungstechnik noch in der Office-Welt bisher ein Standard durchgesetzt hat (Ansätze beinhalten zum Beispiel RFC 1006 und Modbus/TCP). In der Praxis sind derartige nicht standardisierte Protokolle oberhalb von TCP nur durch OPC-Server oder vergleichbare API-Lösungen zu verwenden. Dies ist ein grundlegender Unterschied zum Internet: Internet-Anwendungen erfordern nicht die Verwendung eines bestimmten APIs oder einer bestimmten Komponententechnologie, um Email zu versenden oder Webseiten abzurufen. Der Unterschied ist darin begründet, dass Internet-Anwendungen wohldefinierte, einfache und leicht zu implementierende Anwendungsprotokolle verwenden, welche in der Fabrikautomation gerade fehlen. Ausserdem erfordert die Umstellung einer vorhandenen Kommunikationsinfrastruktur mit gängigen Feldbussen und Punkt-zu-Punkt-Verbindungen auf Ethernet erhebliche Investitionen bei Hardware, Software, Training, Inbetriebnahme und Wartung.

Ein weiteres Verfahren besteht in der Integration eines Web-Servers ins Feldgerät (oft auf Basis eines in das Feldgerät integrierten Industrie-PCs), wodurch das Feldgerät "Internet-fähig" wird. Dieses Verfahren ist allerdings nicht nur sehr kostspielig, sondern hat auch den Nachteil, dass das hierbei verwendete HTML-Dokumentenformat schlecht für die automatisierte Weiterverarbeitung durch einen Softwareprozess geeignet ist.

Das älteste Verfahren zum softwaretechnischen Zugriff auf Automatisierungskomponenten besteht in Gerätetreibern, die ein bestimmtes Industrieprotokoll implementieren. Der Gerätetreiber kann vom Anwendungsprogramm über eine proprietäre Softwareschnittstelle (API) benutzt werden. Die Anwendung selbst braucht wenig oder gar nichts über das verwendete Protokoll zu wissen. Diese Vorgehensweise wurde vorrangig unter DOS verwendet, findet sich aber auch auf Windows- und Unix-Systemen. Die Implementierung des Gerätetreibers erfolgt meist in Form einer statischen oder dynamischen Softwarebibliothek (LIB bzw. DLL) oder in Form von Softwarekomponenten (ActiveX, VCL). Es gibt jedoch keinen API-Standard für den Zugriff auf Automatisierungskomponenten. Ein Anwendungsprogrammierer, der unterschiedliche Industrieprotokolle verwendet (um unterschiedliche Geräte anzusprechen), muss meist mehrere unterschiedliche APIs verwenden. Des weiteren können Gerätetreiber aus praktischer Sicht normalerweise nur von Softwareentwicklern innerhalb selbst programmierter Anwendungen genutzt werden. Die Verwendung mit vorhandenen Standard-Softwareanwendungen (wie zum Beispiel Web-Browsern) oder durch Nicht-Programmierer ist praktisch nicht möglich. Darüber hinaus ist dieser Ansatz nur bedingt netzwerkfähig. Ein API oder Treiber impliziert eine eins-zu-eins-Verbindung zwischen einer Anwendung und einem Treiber bzw. einer DLL. In herkömmlichen Softwarearchitekturen funktionieren solche Verbindungen nur lokal, nicht aber über das Netzwerk. Das bedeutet, dass der Treiber oder die DLL auf demselben Rechner laufen muss wie die Anwendung und die Automatisierungskomponente ebenfalls direkt an diesem Rechner angeschlossen sein muss. Obwohl einige neuere APIs und Komponententechnologien (hauptsächlich DCOM und CORBA) in der Lage sind, "entfernte" oder "verteilte" Funktionen und Prozesse über das Netzwerk aufzurufen, geschieht dies um den Preis erhöhter Komplexität und proprietärer Technologie (im Fall von DCOM). Die grosse Mehrzahl verfügbarer APIs bzw. Treiberschnittstellen (implementiert als DLL oder als DDE-Server) im Bereich der Fabrikautomation ist nicht netzwerfähig; weshalb die Softwareanwendung auf derselben Maschine laufen muss, an den die Automatisierungskomponente direkt angeschlossen ist. Dies nachteilig, da Anwendungen (speziell auf den höheren Ebenen der Automatisierungspyramide) eher zur Zentralisierung tendieren, wogegen Automatisierungskomponenten und Steuerungen eher zur Dezentralisierung tendieren.

DDE (Dynamic Data Exchange) ist ein Verfahren zur Interprozesskommunikation, mit dem Windows-Programme untereinander Nachrichten austauschen können. Windows-Systemaufrufe können dazu benutzt werden, um Nachrichten an eine andere Anwendung zu senden oder Nachrichten von einer anderen Anwendung zu empfangen. Viele Softwarehäuser bieten DDE-Server für Industrieprotokolle an. Oftmals ermöglichen es solche DDE-Server Windows-Anwendungen, Daten mit Automatisierungskomponenten auszutauschen, ohne Details über die verwendeten Übertragungsprotokolle

und -medien zu kennen. Das Problem inkompatibler Protokolle und Datenformate wird durch ein übergeordnetes API umgangen, welches die Details der Datenübertragung vor der Anwendung verbirgt. DDE-Server verwenden ein einheitliches API (das Windows DDE API), welches auch von zahlreichen vorhandenen Windows-Anwendungen (z. B. Microsoft Excel) unterstützt wird. Deshalb können DDE-Server auch von Nicht-Programmierern benutzt werden. Andererseits erfordert DDE das Betriebssystem Microsoft Windows und ist nicht netzwerkfähig. Ausserdem hat DDE eine sehr geringe Performance und einen hohen Overhead.

OPC ist eine Weiterentwicklung der DDE-Server-Technologie für das industrielle Umfeld auf der Basis von OLE (Object Linking and Embedding). OLE ermöglicht es Windows-Anwendungen, untereinander Daten ("Objekte") auszutauschen, selbst per Netzwerk. Die Architektur von OPC ist nahezu identisch mit DDE, mit dem Unterschied, dass ein OPC-Server auch auf einem anderen, per TCP/IP-Netzwerk erreichbarem Rechner installiert werden kann. OPC Client und Server laufen jedoch stets auf PCs mit dem Betriebssystem Microsoft Windows. OPC ist eine komplexe Programmierschnittstelle mit hohem Einarbeitungsaufwand.

Seit kurzem nehmen die Bestrebungen zu, Ethernet als "besseren Feldbus" oder "Feldbus-Ersatz" in der Industrieautomation einzusetzen. Dies aufgrund der Tatsache, dass Ethernet das dominierende Übertragungsmedium im Bereich der Bürokommunikation ist, und Bauteile zu günstigen Preisen von einer Vielzahl von Anbietern bezogen werden können. Ausserdem unterstützt Ethernet höhere Übertragungsgeschwindigkeiten als konventionelle Feldbusse und ermöglicht einen einfachen Internet-Zugang.

Die Ludwigsburger Firma Jetter AG hat daraufhin ein neues Steuerungskonzept entwickelt, welches Ethernet und TCP/IP für den Datenaustausch im gesamten Unternehmen verwendet, bis hin zum Feldgerät in der Produktionsstätte. Dieses Konzept erfordert allerdings den Austausch vorhandener Maschinen und Automatisierungskomponenten, die nicht über eine Ethernet-Schnittstelle verfügen. Das resultierende Netzwerk wird durch Switches und Hochgeschwindigkeits-Ethernet nahezu echtzeitfähig, obwohl Ethernet streng genommen aufgrund seiner nichtdeterministischen Protokollarchitektur nicht für Echtzeitanwendungen geeignet ist. Während Industrieautomation normalerweise in Form einer hierarchischen Architektur dargestellt wird ("Automatisierungspyramide"), ist das Jetter-Modell flach und ohne unterschiedliche hierarchische Ebenen. Es gibt keine isolierten Automatisierungszellen mehr, da die Steuerung zentral vorgenommen wird und nicht dezentral von SPSen. Die zur Steuerung verwendete Software läuft nicht auf einer lokalen SPS, sondern auf einem Server im Netzwerk. Einschränkungen dieses Lösungsansatzes sind: Vorhandene Steuerungs- und Kommunikationshardware muss ersetzt werden durch die Hardware eines einzigen Herstellers; ein Standard-Anwendungsprotokoll (ISO/OSI-Schicht 7) wird nicht spezifiziert, so dass die Anwendungsschicht proprietär ist; somit gibt es auch nur proprietäre, nicht standardisierte Übergänge in die kaufmännische Datenverarbeitung.

ZUSAMMENFASSUNG DER ERFINDUNG

Die vorliegende Erfindung vermeidet die dargestellten Nachteile des gegenwärtigen Stands der Technik mithilfe eines Systems und Verfahrens zum automatisierten Datenaustausch zwischen Steuerungsebene und Planungsebene. Dabei behält die vorliegende Erfindung das hierarchische Modell der Fabrikautomation ("Automatisierungspyramide"), welches sich in vielen tausend Installationen bewährt hat, bei. Ausserdem passen Implementierungen der vorliegenden Erfindung zu vorhandenen standardisierten Softwareschnittstellen. Implementierungen der vorliegenden Erfindung können auf zahlreichen Betriebssystemen und in Feldgeräten vorgenommen werden, ohne dass dafür die Ressourcen eines Windows-PCs erforderlich sind. Des weiteren kann die vorliegende Erfindung unter Benutzung unterschiedlicher Übertragungsmedien und -protokolle implementiert werden.

In bestimmter Hinsicht kann die vorliegende Erfindung ein System zur Verteilung von Automatisierungsdaten (Automation Data Distribution, abgekürzt ADD) verkörpern, bestehend aus einer datenverarbeitenden Station (DS) mit einer Netzwerkadresse, und einer Automatisierungskomponente (AK) mit einer Netzwerkadresse, funktional an die DS gekoppelt; wobei DS und AK so konfiguriert sind, dass sie miteinander durch Verwendung eine oder mehrere Transaktionen kommunizieren, wobei besagte Transaktion aus einem Datagramm besteht, welches vollständig und in Klarschrift codiert ist. In dieser Implementierung können die Datagramme desweiteren ein Quell-Attribut enthalten, welches die Netzwerkadresse des Absenders des Datagramms angibt, ein Ziel-Attribut, welches die Netzwerkadresse des intendierten Empfängers dieses Datagramms enthält, und/oder ein Wert-Attribut mit einem korrespondierenden Einheits-Attribut, welches die Masseinheit des Wert-Attributs angibt. Darüber hinaus können Datagramme ein oder mehrere Sicherheitsattribute enthalten (einschliesslich, jedoch nicht beschränkt auf ein Attribut für eine digitale Signatur und/oder ein Attribut für ein Gültigkeitsende oder "Verfallsdatum"), ein Zeitstempel-Attribut, und/oder ein oder mehrere Fehlersicherungs-Attribute (einschliesslich, jedoch nicht beschränkt auf ein Prüfsummen-Attribut, welches Redundanzinformationen des Datagramm-Inhalts enthält, und ein Sequenznummer-Attribut). Das ADD-System kann darüber hinaus Profile enthalten, welche den Aufbau von einem oder mehreren Datagrammen spezifizieren. Solche Profile können die Konfiguration von mindestens einem besagter Datagramme für die Benutzung mit einer bestimmten AK spezifizieren. Das oder die Datagramme können so gestaltet sein, dass ein Zustand der AK repräsentiert wird. Das oder die Datagramme können so gestaltet sein, dass ein Kommando von einer DS an eine AK repräsentiert wird. Das oder die Datagramme können dafür benutzt werden, den Empfang eines vorherigen Datagramms zu quittieren. Die Transaktionen können aktiv, passiv, oder eine Kombination von aktiv und passiv sein. Optional kann das ADD-System eine Koordinierungsstelle beinhalten, welche funktional mit den DS und AK gekoppelt ist und welche den DS und AK Netzwerkadressen zuweisen kann. In einigen Implementierungen können die Datagramme das FactoryXML-Format verwenden. In anderen Implementierungen können die Klarschriftbeschreibungen das UTV-Format verwenden.

In anderer Hinsicht kann die vorliegende Erfindung ein System zur Verteilung von Automatisierungsdaten darstellen, welche eine Planungsebene mit einem Planungs-Netzwerk umfasst, an das eine oder mehrere DS angeschlossen sind; eine Steuerungsebene mit einem Steuerungs-Netzwerk, an das ein oder mehrere AK angeschlossen sind; und ein oder mehrere Gateways, die funktional mit der oder den AK und mit dem Planungsnetzwerk gekoppelt sind, wobei das oder die Gateways dazu dienen, der oder den DS die Kommunikation mit der oder den AK zu erleichtern, indem sie ein oder

mehrere Datagramme erzeugen und an besagte AK schicken; und in dem die AK so arbeiten kann bzw. können, dass sie mit der oder den DS dadurch kommunizieren, dass sie eine oder mehrere Datagramme durch das oder die Gateways an die DS generieren und senden.

In anderer Hinsicht kann die vorliegende Erfindung ein Datagramm umfassen, welches innerhalb eines Systems zur Verteilung von Automatisierungsdaten verwendet wird, welches wiederum aus mindestens einer AK besteht, die funktional mit mindestens einer Zielkomponente gekoppelt ist, wobei wobei das Datagramm aus einem Quell-Attribut besteht, das die Quelle (den Absender) des Datagramms kennzeichnet, einem Ziel-Attribut, welches das Ziel (den Empfänger) des Datagramms kennzeichnet, wobei das Ziel eine DS oder eine AK sein kann, und das Datagramm vollständig codiert ist.

In wiederum anderer Hinsicht kann die vorliegende Erfindung ein Verfahren zur Verteilung von Automatisierungsdaten beinhalten in einem System, welches aus einer DS besteht, die funktional mit einer AK gekoppelt ist, wobei das Verfahren eine Quelleinheit beinhaltet, die ein Datagramm generiert, welches aus vollständig codierten Beschreibungen in Klarschrift besteht, und einer Zieleinheit, welche das Datagramm empfängt und darauf in geeigneter Weise reagiert. In dieser Implementierung kann die Quelleinheit eine DS sein, die ein Abfrage-Datagramm an die Zieleinheit (eine AK) sendet, und die Zieleinheit dadurch antwortet, dass sie ein Antwort-Datagramm an die Quelleinheit schickt, wobei das Antwort-Datagramm aus einem Wert-Attribut und einem zugehörigen Attribut mit der verwendeten Masseinheit besteht. Weiterhin kann das Verfahren in dieser Implementierung eine Quelleinheit (eine AK) beinhalten, die aktiv beim Eintreten vordefinierter auslösender Ereignisse ein oder mehrere Datagramme erzeugt und das oder die Datagramme an die Zieleinheit sendet.

In der vorliegenden Patentschrift bedeutet der Begriff "ein" "ein oder mehrere".

KURZBESCHREIBUNG DER ZEICHNUNGEN

Die folgenden Zeichnungen sind Bestandteil der vorliegenden Patentanmeldung und sind beigefügt, um bestimmte Aspekte der vorliegenden Erfindung zu veranschaulichen. Durch den Verweis auf eine oder mehrere der Zeichnungen in Verbindung mit der detaillierten Beschreibung spezifischer Implementierungen, wie sie hier dargestellt werden, ist die Erfindung eventuell besser zu verstehen.

Fig. 1 ist ein abstraktes Blockdiagramm einer beispielhaften Implementierung der vorliegenden Erfindung, welches ein System zur Verteilung von Automatisierungsdaten veranschaulicht.

Fig. 2 ist ein abstraktes Blockdiagramm einer beispielhaften Implementierung der vorliegenden Erfindung und veranschaulicht ein alternatives System zur Verteilung von Automatisierungsdaten, welches ein Hardware-Gateway beinhaltet.

Fig. 3 ist ein abstraktes Blockdiagramm einer beispielhaften Implementierung der vorliegenden Erfindung und veranschaulicht ein alternatives System zur Verteilung von Automatisierungsdaten, welches ein DDE-Software-Gateway beinhaltet.

Fig. 4 ist ein abstraktes Blockdiagramm einer beispielhaften Implementierung der vorliegenden Erfindung und veranschaulicht ein alternatives System zur Verteilung von Automatisierungsdaten, welches ein OPC-Software-Gateway beinhaltet.

BESCHREIBUNG BEISPIELHAFTER IMPLEMENTIERUNGEN

Fig. 1 zeigt ein System zur Verteilung von Automatisierungsdaten ("ADD") entsprechend einer beispielhaften Implementierung der vorliegenden Erfindung. Ein Vorteil der vorliegenden Erfindung besteht darin, dass sie den Datenfluss zwischen und innerhalb der Steuerungsebene (**160**) und der Planungsebene (**150**) eines geschlossenen Systems wie einer Fabrik oder eines Büros erleichtert. Kaufmännische Softwareanwendungen auf Computern befinden sich normalerweise innerhalb der Planungsebene (**150**), während sich speicherprogrammierbare Steuerungen, Peripheriegeräte, Feldgeräte und automatisierungstechnische Softwareanwendungen normalerweise innerhalb der Steuerungsebene befinden. Für die Zwecke der vorliegenden Schrift wird die Bezeichnung "datenverarbeitende Station" (DS) für steuernde, überwachende und auswertende Geräte und Softwareanwendungen sowohl in der Steuerungsebene als auch in der Planungsebene verwendet. Der Begriff "Automatisierungskomponente" (AK) bezieht sich hingegen auf Peripheriegeräte, Feldgeräte und automatisierungstechnische Softwareanwendungen. Systemkomponenten (sowohl AK als auch DS) können physikalische Geräte sein, Peripheriegeräte und Softwareanwendungen.

Die vorliegende Erfindung ermöglicht wirkliche Datenverteilung innerhalb des Gesamtsystems durch die Verwendung eines einheitlichen Transaktions-Formats, das von allen Systemkomponenten verwendet wird, egal auf welcher Ebene sie sich befinden. Die Transaktions-Grundeinheit für Datenübertragung in ADD ist ein vollständig codierter Gerätezustand oder ein vollständig codiertes Kommando. Vollständig codiert bedeutet, dass keine zusätzliche Information (wie eine vordefinierte Umsetzungstabelle) erforderlich ist, um den Gerätezustand oder das Kommando zu bestimmen. Im Gegensatz hierzu liefern gängige AK oft nur einen einzelnen Wert, wenn sie abgefragt werden. Zum Beispiel kann eine gängige Waage ihre Daten über ihre Computerschnittstelle einfach mit einer Zeilenbegrenzung versehen abliefern (z. B. 525.55[CR]). Obwohl 525.55 auf eine Gewichtsmessung hinweisen kann, müsste eine DS noch wissen, welche Masseinheit die betreffende Waage verwendet hat. In einer typischen Installation würde die DS "wissen" (vielleicht durch Heranziehung einer Umsetzungstabelle), dass die Daten von einer bestimmten AK (zum Beispiel "Waage 3") in Kilogramm kalibriert sind. Ohne Kenntnis der verwendeten Masseinheit sind Daten von AK von begrenztem Wert. Selbst bei Verwendung von Umsetzungstabellen ist diese Art der Implementierung schlecht geeignet für eine Verteilung der Daten innerhalb eines Netzwerks, da eine beliebige DS innerhalb des Netzwerks nicht in der Lage wäre, die Datenquelle, den Zeitpunkt und die Masseinheit der Messung zu bestimmen. Ausserdem wären bei Konfigurationsänderungen netzwerkweite Updates der Umsetzungstabellen erforderlich. ADD begegnet diesem Problem durch die Verwendung von vollständig codierten Gerätezuständen oder Kommandos, welche im vorangegangenen Beispiel die Masseinheit des übertra-

genen Wertes beinhalten würden.

Ein vollständig codierter Datensatz sollte einer DS wenigstens ermöglichen, ohne Zugriff auf externe Information die Datenquelle zu ermitteln und, falls Daten übertragen werden, die Einheit dieser Daten. Bei vollständiger Codierung sind Umsetzungstabellen beim Empfänger nicht nötig. Die übermittelte Information sollte von beliebigen Empfängern ohne weitere Anforderungen verarbeitet werden können.

In bevorzugten Implementierungen verwendet die vorliegende Erfindung Datensätze in Klartext, wann immer möglich und sinnvoll. Die Codierung von Daten in Klarschrift hat mehrere Vorteile gegenüber anderen Codierungsarten. Erstens ist sie menschlich lesbar. Das allereinfachste "Terminal" (das heisst, ein Drucker oder ein einfaches Datensichtgerät) kann benutzt werden, um ADD-Daten für einen menschlichen Benutzer anzuzeigen. Viele altgediente Wartungsingenieure im Aussendienst hat es viel Zeit und Frustration gekostet, gestörte Übertragungskanäle durch das Entziffern von Bitfeldern, Prüfsummen usw. zu analysieren. Die Möglichkeiten zur Fehlersuche sind bei ADD hervorragend, was wichtig für die Fälle ist, in denen aufwendige Datentester normalerweise nicht zur Verfügung stehen und Ausfallzeiten teuer sind. Mit ADD kann ein Wartungsingenieur ein Gerät sogar mit einem einfachen Terminal-Emulator manipulieren, da ADD-Datensätze ohne aufwendige Software zusammengestellt und übertragen werden können.

Desweiteren ist die von ADD verwendete Klarschrift-Codierung weniger fehleranfällig für Versionskonflikte innerhalb einer verteilten Umgebung. Bei Binärcodierung müssen Umsetzungstabellen verwendet werden, um die übertragenen Daten zu interpretieren. Wenn die Umsetzungstabellen lokal gespeichert werden (wie dies normalerweise der Fall ist), treten Fehler auf, wenn diese Tabellen nicht im gesamten Netzwerk upgedated werden. Ein Vorteil von ADD liegt darin, dass die Notwendigkeit von Umsetzungstabellen zur Interpretation von Daten weitgehend schwindet.

Weiterhin stellt die Verwendung von Klarschrift-Codierung wenig Ansprüche an den Übertragungskanal. In einigen Implementierungen der vorliegenden Erfindung kann eine 7-Bit-Codierung verwendet werden, und ASCII-Steuerzeichen können für Protokollzwecke verwendet werden, wenn sie nicht in den Nutzdaten auftreten. Diese Implementierungen vereinfachen die Konstruktion von Gateways. Die vereinfachte Datenformatierung vereinfacht die Verteilung der Daten bedeutend.

Wie gezeigt kann eine Implementierung der vorliegenden Erfindung ein oder mehrere DS (kaufmännische Softwareanwendung **[100]** und speicherprogrammierbare Steuerung / SPS **[110]**) und eine oder mehrere AK (**120** und **130**) beinhalten, welche funktional durch ein Übertragungsmedium (**140**) miteinander gekoppelt sind. Obwohl DS **100** und **110** und AK **120** und **130** als direkt an das Übertragungsmedium angeschlossen dargestellt sind, wird ein technisch kundiger Leser verstehen, dass "funktional gekoppelt" keine direkte Verbindung erfordert. Für die Zwecke der vorliegenden Patentschrift kann ein Gerät als funktional gekoppelt mit einem anderen Gerät bezeichnet werden, wenn die Geräte miteinander Informationen austauschen können, unabhängig vom Vorhandensein irgendwelcher zwischengeschalteter Geräte oder Übertragungsmedien.

In einer bevorzugten Implementierung kann ADD direkt in jede AK implementiert werden. Dadurch könnten AK **120** und **130** direkt mit DS **100** und **110** kommunizieren. Diese Implementierung wäre geeignet, traditionelle Industrieprotokolle wie zeilenorientiertes ASCII, Modbus oder das AK-Protokoll (ein Anwendungsprotokoll aus der Automobil-Zulieferindustrie) zu ersetzen. TCP/IP über Ethernet ist ein bevorzugtes Übertragungsmedium **140**. Ein technisch kundiger Leser wird indessen verstehen, dass jedes geeignete Übertragungsmedium **140** verwendet werden kann, einschliesslich (jedoch nicht beschränkt auf) serielle Asynchronverbindungen, ISDN, drahtloses TCP/IP, USB, Verbindungen über Parallelschnittstellen und GSM.

In einigen Fällen mögen Systemkomponenten der Steuerungsebene (**160**) nicht in der Lage sein, die erforderlichen Netzwerkprotokolle zur direkten Kommunikation mit Systemkomponenten auf der Planungsebene (**160**) zu benutzen oder sich daran anzuschliessen. In diesen Fällen kann ein Gateway benutzt werden, um eine Kommunikation zu ermöglichen. Viele der heute gebräuchlichen Gateways (sowohl Hardware- als auch Softwaregateways) können hierfür benutzt werden. Ein technisch kundiger Leser könnte angeben, welches spezielle Gateway mit den Systemkomponenten in einer bestimmten Implmentierung benutzt werden kann.

Wie in **Fig.** 2 dargestellt, können ADD-Hardware-Gateways (**210, 220, 230**) benutzt werden, wo vorhandene AK nicht in der Lage sind, die erforderlichen Netzwerkprotokolle zu benutzen oder sich daran anzuschliessen. Ein ADD-Hardware-Gateway übersetzt vom Netzwerk-Medium zum Punkt-zu-Punkt-Medium (beispielsweise serielle Asynchronschnittstelle) und kümmert sich um die erforderlichen Protokollumwandlungen auf den höheren Schichten (wie zum Beispiel der Präsentationsschicht und der Anwendungsschicht). ADD erfordert im allgemeinen keine grosse Rechen- und Speicherkapazität. ADD-Hardware-Gateways können daher in bevorzugten Implementierungen as kleine Protokollkonverter implementiert werden, die hutschienenmontiert oder als Zwischenstecker auf eine vorhandene Kommunikationsschnittstelle der AK aufgesteckt werden können. Ein Beispiel für ein ADD-Hardware-Gateway ist (ohne hierauf beschränkt zu sein) ein SMS-Gateway.

In anderen (nicht abgebildeten) Implementierungen der vorliegenden Erfindung kann das Übertragungsmedium **140** aus zwei Netzwerken bestehen, einem Netzwerk der Steuerungsebene und einem Netzwerk der Planungsebene, mit einem oder mehreren Hardware-Gateways zur Erleichterung der Kommunikation zwischen den beiden Ebenen. Das Planungsnetzwerk und das Steuerungsnetzwerk können beide als Teile des Gesamtnetzwerks **140** betrachtet werden. In anderen Implementierungen mögen die AK **120** und **130** überhaupt nicht direkt mit dem Übertragungsmedium **140** verbunden sein. In diesen Implementierungen können die AK **120** und **130** stattdessen mit einer SPS **110** verbunden sein, welche ihrerseits an das Übertragungsmedium **140** über ein Hardware- oder Software-Gateway (siehe unten) gekoppelt ist.

Alternativ kann die Gateway-Funktionalität auch softwaremässig implementiert werden. Wie in **Fig.** 3 dargestellt, kann ein ADD-Softwaregateway als DDE Client **300** implementiert werden, der mit AK **340** über einen DDE Server **320** kommuniziert. Wie dargestellt, kann das ADD Gateway auf dem Computer installiert werden, auf dem der DDE Server läuft. DDE-Implementierungen laufen gegenwärtig ausschliesslich auf Windows-PCs. Da die meisten Windows-PCs netzwerkfähig sind, hat diese Implementierung den zusätzlichen Vorteil, dass ADD-Systemkomponenten im gesamten Netzwerk den DDE Server benutzen und mit ihm kommunizieren können. Der DDE Server **320** kann mit einer AK **340** über ein Industrieprotokoll **330** kommunizieren und dieses in DDE API-Aufrufe **310** für die Kommunikation mit dem

DDE Client **300** übersetzen, und umgekehrt.

Wie in **Fig. 4** gezeigt, kann ein ADD Softwaregateway auch als OPC Software-Gateway implementiert werden. Im speziellen kann ein ADD Gateway als OPC Client-Anwendung **400** implementiert werden. In der abgebildeten Implementierung meldet sich das ADD Gateway als OPC Client bei einem OPC Server **430** an und setzt ADD in OPC **400** und umgekehrt um, um mit AK **450** zu kommunizieren. Ein Vorteil hiervon ist, dass vorhandene OPC Client-Software **410** weiter benutzt werden kann, da OPC Server mehrere Clients gleichzeitig bedienen können.

Eine dritte Gruppe von Gateway-Software (ohne Abbildung) unterstützt jene Industrieprotokolle, die von sich aus TCP/IP als Übertragungskanal verwenden, wie RFC 1006, oder die Netzwerkvariante von Modbus, oder Peripheriegeräte wie digitale I/O-Baugruppen mit proprietären Netzwerkprotokollen. Diese Art ADD-Gateway kann auf irgendeinem Computer im Netzwerk installiert werden. Eine DS würde in diesem Fall nicht direkt mit der Systemkomponente "hinter" dem Gateway kommunizieren. Stattdessen würde die DS ein Datagramm an die Netzwerkadresse des Gateways schicken. Das Gateway übersetzt das Datagramm in die Protokolle und Kommandos der Systemkomponente und umgekehrt.

In anderen Implementierungen (ohne Abbildung) kann eine Kombination aus Software- und Hardware-Gateways innerhalb desselben Systems verwendet werden. Desweiteren können einige Implementierungen der vorliegenden Erfindung Gateways für Systemkomponenten verwenden, die Gateways erfordern, und zusätzlich andere Systemkomponenten, die ADD-kompatibel sind und keine Gateways erfordern.

Transaktionen

In ADD-Systemen erfolgt die Kommunikation zwischen AK und DS (und untereinander) durch eine oder mehrere Transaktionen. Allgemein gesprochen sind Transaktionen ein Informationsaustausch zwischen verschiedenen Systemkomponenten des ADD-Systems in Form vollständig codierter Datagramme. Für Zwecke der vorliegenden Patentschrift bestehen alle Datagramme aus vollständig codierter Information. In einigen Fällen kann der Inhalt eines Datagramms Daten in einem Format enthalten, das nicht ohne weiteres menschlich lesbar ist (zum Beispiel binär oder hexadezimal codierte Daten). Bevorzugterweise sind Datagramme jedoch menschlich lesbar, wenn immer es möglich ist.

Vollständig codierte Datagramme sind die Basiseinheit einer ADD-Transaktion. Ein Datagramm kann mit einem "Record" (Datensatz) einer Datenbank verglichen werden. Ein Datagramm kann unterschiedliche Funktionen haben. Typische Funktionen sind (ohne hierauf beschränkt zu sein): Darstellung eines Gerätezustands, Darstellung eines Kommandos, oder Quittierung eines vorangegangenen Kommandos. In einigen Fällen kann ein Datagramm mehrere Funktionen gleichzeitig haben.

Im Hinblick auf AK können Transaktionen in die beiden Kategorien "aktiv" und "passiv" unterteilt werden. Bei passiven Transaktionen empfängt eine AK ein Kommando oder eine Anfrage von einer anderen Systemkomponente und kann darauf, falls erforderlich, entsprechend antworten. Bei aktiven Transaktionen kann eine AK Daten spontan, auf eigene Initiative hin übertragen (normalerweise beim Eintreten eines auslösenden Ereignisses, oder zyklisch). In vielen Systemen sind aktive Transaktionen zu bevorzugen, da sie den Trend zur Dezentralisierung verstärken – sie reduzieren den Datenverkehr im Netz und vereinfachen die Programmierung von datenverarbeitenden Anwendungen, dem ereignisgesteuerten Paradigma der modernen Softwareentwicklung entsprechen, wie es für IT-Anwendungen üblich ist. Die spezifische Struktur und Konfiguration jeder Transaktion kann von mehreren Faktoren abhängig sein, inklusive (jedoch ohne hierauf beschränkt zu sein) der Art der Implementierung, dem Datenformat (siehe unten) und den Übertragungskanälen, die in einem ADD-System verwendet werden.

Jedes Datenformat kann benutzt werden, das zu vollständig codierten Datagrammen führt. Dabei werden zwei Datenformate – FactoryXML und UTV (Unstructured Tag/Value) bevorzugt, weil sie gut zu existierenden Softwareschnittstellen passen. FactoryXML ist abgeleitet vom Format der extensible Markup Language (XML). UTV enthält ähnliche Inhalte wie FactoryXML, verwendet jedoch eine einfachere, unstrukturierte Syntax.

FactoryXML

FactoryXML benutzt die im XML-Standard definierte Syntax, entspricht aber nicht notwendigerweise dem XML-Dokumentenmodell. Anstelle des für XML-Dokumente typischen statischen Inhalts kann (und wird) sich der Inhalt eines FactoryXML-Datagramms, welches einen Gerätezustand repräsentiert, dynamisch ändern. Ausserdem können sich mehrere FactoryXML-Datagramme auf ein und denselben Gegenstand beziehen. Dies ist ein wichtiger Unterschied zwischen XML und FactoryXML. In XML ist die Grundeinheit ein Dokument (oder eine Datei). Unterschiedliche Dokumente repräsentieren unterschiedliche Dinge (Bücher, Artikel usw.). In FactoryXML ist die Grundeinheit ein Datagramm. Unterschiedliche FactoryXML-Datagramme können sich auf ein und denselben Gegenstand beziehen (zum Beispiel auf ein bestimmtes Peripheriegerät), und repräsentieren dabei unterschiedliche Zustände zu unterschiedlichen Zeitpunkten. Obwohl sich der Zustand des Peripheriegeräts ändern kann, bleibt das Peripheriegerät doch ein und dasselbe. Da ein vollständig codiertes ADD-Datagramm einen Gerätezustand eindeutig repräsentieren muss, darf es innerhalb eines FactoryXML-Datagramms keine mehrfachen identischen Unterelemente geben. Ein Sensor kann zum Beispiel zu einem bestimmten Zeitpunkt immer nur einen bestimmten Wert liefern. Jede Änderung muss in einem separaten Datagramm codiert werden. Ein Sensor-Datagramm mit mehrfachen "value"-Elementen ist daher unzulässig (wogegen mehrfache "value"-Elemente wie "value1", "value2" usw. bei komplexen Sensoren benutzt werden können, solange keine mehrfachen Instanzen eines einzelnen Elements existieren). Demgegenüber sind mehrfache Unterelemente in XML-Dokumenten (z. B. "Kapitel", "Absatz") zulässig und sogar üblich. Trotz der Unterschiede zwischen XML und FactoryXML können FactoryXML-Datagramme von gewöhnlichen XML-Parsern verarbeitet werden. Deshalb kann eine XML-fähige Datenbank FactoryXML-Datagramme auch unmittelbar speichern.

Ein beispielhaftes FactoryXML-Datagramm kann wie folgt aussehen:

```
<sensor src="Themometer2" t="181234"
seq = "42537" crc="5528">
        <value>45.755</value>
        <unit><Fahrenheit/></unit>
</sensor>
```

Implementierungen der vorliegenden Erfindung, welche FactoryXML-Datagramme (wie das obige Beispiel) verwenden, können aus einem oder mehreren Elementen bestehen, die einen Elementnamen ("Tag") und einen optimalen Inhalt (Wert) haben.

In einigen Implementierungen können Tags lediglich aus Buchstaben in Kleinschreibung bestehen, mit Ausnahme vordefinierter Tags, bei denen der erste Buchstabe gross geschrieben ist. Alternativ kann Gross- und Kleinschreibung ignoriert werden.

Ein FactoryXML-Element kann Unterelemente beliebiger hierarchischer Tiefe enthalten. Unterelemente des FactoryXML-Wurzelelements müssen jedoch eindeutig sein; mehrfache identische Unterelemente sind nicht erlaubt, egal, ob sie auf derselben oder auf unterschiedlichen hierarchischen Ebenen stehen. Tags für FactoryXML-Elemente können frei definiert werden, mit der Ausnahme vordefinierter Standard-Tags (siehe unten).

FactoryXML beinhaltet eine standardisierte Attributliste. Welche Standardattribute in einem bestimmten FactoryXML-Datagramm verwendet werden, kann konfiguriert werden (zum Beispiel durch die Verwendung von Profilen, siehe unten). Die Standard-Attributliste kann folgende Attribute enthalten (ohne hierauf beschränkt sein zu müssen): Attribute zur Datensicherheit (zum Beispiel eine digitale Signatur, ein Gültigkeitsende usw.), einen Zeitstempel, und Attribute zur Sicherung der Datenintegrität (zum Beispiel eine Prüfsumme, CRC usw.).

Obwohl FactoryXML-Attribute generell optional sind, kann ihre Verwendung oder Nicht-Verwendung in bestimmten Situationen vorgeschrieben sein, die durch die Konfiguration des ADD-Systems spezifiziert werden. So kann zum Beispiel die Verwendung einer digitalen Signatur die gleichzeitige Verwendung eines Zeitstempels und einer Sequenznummer erfordern. Ein FactoryXML-kompatibles Gerät muss in der Lage sein, alle Standard-Attribute zu verarbeiten und darf nicht mit Fehlerzuständen oder undefiniertem Verhalten auf den Empfang irgendeines Standard-Attributs reagieren.

In vielen FactoryXML-Implementierungen kann die Codierung von Datagrammen im ASCII-Zeichensatz erfolgen. Numerische Daten können als Fliesskommawerte mit einem Dezimalpunkt codiert werden, mit einem optionalen Vorzeichen (Präfix) und einem optionalen Exponenten (Suffix). Ein weggelassenes Vorzeichen kann einen positiven Wert anzeigen. Ein weggelassener Exponent kann einen Exponenten von eins anzeigen. Daten können auch als ganzzahlige Werte (Integer) dargestellt werden, entweder in dezimaler oder in hexadezimaler Schreibweise. Hexadezimale Schreibweise kann durch den Präfix "0x" angezeigt werden. Negative dezimale Werte können durch ein vorangestelltes Minuszeichen kenntlich gemacht werden. Ein weggelassenes Vorzeichen kann einen positiven Wert kennzeichnen. Daten können auch als Textwerte dargestellt werden. Textwerte setzen sich aus Zeichenketten zusammen, wie sie im XML-Dokumentenformat spezifiziert sind. Die Verarbeitung von Textzwischenräumen (Whitespace) kann identisch zum XML-Dokumentenformat erfolgen. Schliesslich kann FactoryXML einen Satz vordefinierter Tags zur eindeutigen Kennzeichnung bestimmter Zustände und Einheiten bereitstellen. Vordefinierte Tags können in FactoryXML-Datagrammen als leere Elemente verwendet werden. Der Anfangsbuchstabe eines vordefinierten Tags ist immer gross geschrieben. Beispiele umfassen (ohne Anspruch auf Vollständigkeit): <On/>, <Off/>, <Leftl>, <Right/>, <Up/>, <Down/>, <Open/>, <Closed/>, <Full/>, <Empty/>. Die Verwendung dieser Tags kann ohne die Angabe einer Einheit erfolgen. Für andere Situationen kann es vordefinierte Tags für alle Basiseinheiten und abgeleitete Masseinheiten geben, wie beispielsweise (ohne hierauf beschränkt zu sein): <Meter/>, <Kilogram/>, <Second/>, <Ampere/>, <Kelvin/>, <Mol/>, <Candela/>, <Fahrenheit/>, <Celsius/>, <Inch/>, <Ampereh/>, <Percent/>.

In einer bevorzugten Implementierung verwenden Anwendungen vordefinierte FactoryXML-Tags wenn immer möglich, um die Möglichkeit nicht eindeutiger Wertebezeichnungen zu vermindern. Wo Werte bzw. Masseinheiten nach Gutdünken als Textfelder codiert werden, gibt es Interpretationsbedarf. Beispiel: Das folgende FactoryXML-Datagramm –

```
<sensor src="Themometer2" t="181234">
        <value>45.755</value>
        <unit>F</unit>
</sensor>
```

– verwendet keine standardisierte Einheit. Der Wert "F" für das Element <unit> legt die Einheit Fahrenheit nahe, ist jedoch nicht eindeutig. Das folgende FactoryXML-Datagramm eliminiert jede Mehrdeutigkeit durch die Verwendung eines vordefinierten Tags:

```
<sensor src="Themometer2" t="181234">
    <value>45.755</value>
    <unit><Fahrenheit/></unit>
</sensor>
```

In einigen Implementierungen können spezifische oder besonders aussagekräftige Tags verwendet werden, um Gerätezustände zusätzlich oder anstelle des <value>-Tags zu repräsentieren. So kann zum Beispiel für ein Thermometer auch ein Tag namens <temperature> verwendet werden. In ähnlicher Weise kann in einem Durchflussmesser ein Tag namens <rate> verwendet werden. Einige Einheiten können mehrere Werte übermitteln und hierfür unterschiedliche Tags verwenden (zum Beispiel <rate1>, <rate2> usw.).

Ein "src"-Attribut kann verwendet werden, um die Datenquelle (d. h. die Netzwerkadresse der Datenquelle) eines FactoryXML-Datagramms kenntlich zu machen. Im Hinblick auf die vorliegende Erfindung ist mit "Netzwerkadresse" die eindeutige Adresse gemeint, die den eindeutigen Ort einer Einheit innerhalb des Systems beschreibt. Eine Netzwerkadresse kann eine IP-Adresse, IPX-Adresse usw. sein, oder es kann eine symbolische Adresse sein (zum Beispiel "Thermometer2"). Quelladressen können frei definiert werden, um ein bestimmtes Gerät oder eine bestimmte Anwendung innerhalb eines Netzwerkes zu identifizieren. In einer bevorzugten Implementierung ist die Quelladresse ein verständlicher symbolischer Name. ADD-Quelladressen müssen eindeutig sein, aber müssen nicht identisch mit Netzwerkadressen des Übertragungsmediums sein (d. h. Hostnamen oder IP-Adressen); sie müssen auch nicht für ein bestimmtes Gerät identisch sein, da auf einem Gerät mehrere Anwendungen laufen können, die dann unterschiedliche Adressen benötigen.

Darüber hinaus unterstützt FactoryXML weltweit eindeutige Quelladressen, die von einer Koordinierungsstelle vergeben werden können. Solch eine Adresse kann zum Beispiel in ein Peripheriegerät vom Hersteller "eingebrannt" werden, so dass dieses Gerät ohne weitere Konfigurationsschritte bei Inbetriebnahme adressiert werden kann.

Das "src"-Attribut ermöglicht ein sehr einfaches und genaues Mitprotokollieren von Transaktionen, da Datagramme sofort einem Absender zugeordnet werden können. Darüber hinaus ermöglicht es die Verwendung eines Transportkanals für mehrere Geräte (Multiplexing).

In bevorzugten Implementierungen verwendet jedes FactoryXML-fähige Gerät eine weltweit eindeutige Quelladresse, die von der Koordinierungsstelle ausgegeben wird. Wo dies nicht möglich ist, kann eine andere voreingestellte Quelladresse (wie zum Beispiel "ADD") verwendet werden. In diesem Fällen kann die voreingestellte Quelladresse verwendet werden, um ein Gerät zu initialisieren, so dass Anwendungen die Möglichkeit haben, festzustellen, ob das Gerät ADD-kompatibel ist. In anderen Implementierungen kann die Verwendung einer voreingestellten Quelladresse unterbleiben.

Ein Datagramm kann weitere Informationen enthalten, wie in folgendem Beispiel:

```
<set dest="valve3" sig="0x420c4eff84ed96ac420c4eff84ed96ac">
<actuator src="scada" t="20000101T034500" seq="42537"
exp="20000101T110323.45Z">
        <value><On/></value>
</actuator>
</set>
```

In diesem Datagramm bezeichnet das Attribut für die Zieladresse ("dest") die Datensenke, an die das Datagramm gesendet wird. Die Zieladresse ist identisch mit der Quelladresse des Geräts, an das das Datagramm gesendet wird. Es kann vordefinierte globale Adressen geben, wie "*" oder "any", die verwendet werden können, wenn ein Datagramm an alle Geräte bzw. Anwendungen gesendet werden soll, die über einen gegebenen Übertragungskanal erreicht werden können. In einigen Implementierungen können im Zieladress-Attribut auch Platzhalter (Wildcards) verwendet werden.

Ein Zeitstempel-Attribut ("t" im obigen Beispiel) kann dazu verwendet werden, den Zeitpunkt zu spezifizieren, zu dem das Datagramm erzeugt wurde. In bevorzugten Implementierungen haben alle Datagramme einen Zeitstempel. Die Verwendung von Zeitstempeln ist besonders sinnvoll in verteilten Umgebungen, bei denen Daten ohne weitere Zwischenschritte direkt in eine Datenbank geschrieben werden. Für die Codierung von Zeitstempeln können verschiedene Methoden benutzt werden. In einer bevorzugten Implementierung kann der ISO-6801-Standard benutzt werden, um einen Zeitstempel mit Stunden, Minuten und Sekunden ohne Doppelpunkte und Bindestriche zu erzeugen. Ausserdem können Jahr, Monat, Tag, Millisekunden und Zulu-Zeit (UTC) angegeben werden. Für netzwerkfähige Geräte kann die Uhrzeit mit dem in RFC 1129 spezifizierten Protokoll synchronisiert werden. Für Geräte, die die Uhrzeit nicht berechnen können, kann ein voreingestellter Wert verwendet werden. In anderen Implementierungen sind Zeitstempel optional.

Ein Sequenz-Attribut ("seq") kann verwendet werden, um die Reihenfolge eines bestimmten Datagramms in bezug zu vorherigen und nachfolgenden Datagrammen kenntlich zu machen. In bevorzugten Implementierungen kann die Sequenznummer als positiver ganzzahliger Wert zwischen 0 und 65535 geführt werden. Bei Überlauf kann die Sequenznummer wieder auf 0 zurückgesetzt werden. Das Sequenz-Attribut ermöglicht es, Datenverluste und Sequenzfehler zu erkennen, die bei nicht-fehlerkorrigierenden Übertragungskanälen (wie beispielsweise UDP) auftreten können. Bei der Verwendung von Quittungen (siehe unten) kann dadurch ein Datenverlust durch erneute Übertragung verhindert werden. Beim Übertragen von Kommandos kann die datenverarbeitende Station die Beantwortung des Kommandos eindeutig re-

ferenzieren. Wenn die datenverarbeitende Station eine Sequenznummer zusammen mit einem Kommando (set) oder einer Abfrage (poll) verwendet, kann die Antwort dieselbe Sequenznummer verwenden.

In einigen Implementierungen kann ein Prüfsummen-Attribut ("crc") verwendet werden, welches Redundanzinformation des Datagramm-Inhalts enthält. Die Prüfsumme ermöglicht eine Fehlererkennung bei Verwendung ungesicherter Übertragungskanäle, wie zum Beispiel seriellen Asynchronschnittstellen. In bevorzugten Implementierungen wird das Prüfsummenattribut im Quittungsbetrieb verwendet. Eine Prüfsumme zusammen mit Quittungsdatagrammen kann eine automatische Fehlerkorrektur bereitstellen. So kann ein Empfänger zum Beispiel anhand der Prüfsumme feststellen, ob das Datagramm fehlerfrei übertragen wurde. Falls nicht, kann er eine erneute Übertragung durch Absenden einer negativen Quittung erzwingen. Diese Prozedur wird wiederholt, bis das Datagramm fehlerfrei übertragen wurde (d. h., bis Inhalt und Prüfsumme zusammen passen). Dies ist nicht besonders sinnvoll für TCP/IP-Übertragungskanäle, es kann aber wichtig für serielle Punkt-zu-Punkt-Verbindungen und Modemstrecken sein, wo eine automatische Fehlerkorrektur nicht zwingend bereits durch den Übertragungskanal vorgenommen wird. Die Implementierungsverfahren für Prüfsummen sind allgemein bekannt. In einer Implementierung der vorliegenden Erfindung kann die Prüfsumme als CRC-16 für den Datagramm-Inhalt ohne Leerzeichen (Whitespace) und ohne die Prüfsumme selbst berechnet werden.

Ein Attribut für eine digitale Signatur ("sig") kann verwendet werden, welches verschlüsselte Redundanzinformation des Datagramm-Inhalts enthält. In einer bevorzugten Implementierung kann die Signatur aus dem Datagramm-Inhalt mit dem RSA-Algorithmus berechnet werden, welcher zuvor mit der MDS-Hash-Funktion codiert wird. Ein Public-Key-Verfahren kann verwendet werden. Der Empfänger kann den öffentlichen Schlüssel des Senders verwenden, um den verschlüsselten Hash-Wert zu entschlüsseln und mit einem zweiten, selbst aus dem Datagramm-Inhalt berechneten Hash-Wert zu vergleichen. Wenn die beiden Werte übereinstimmen, wird der Inhalt des Datagramms als gültig betrachtet. In einer bevorzugten Implementierung müssen bei Verwendung einer digitalen Signatur auch eine Quelladresse, ein Zeitstempel und eine Sequenznummer verwendet werden. Zur weiteren Erhöhung der Sicherheit kann ein Gültigkeitsende ("exp") verwendet werden. Ein Datagramm mit einer digitalen Signatur darf nicht als Antwort auf ein Datagramm ohne Signatur oder als Antwort auf ein Datagramm mit einer ungültigen Signatur gesendet werden. Bei der Verwendung von Netzwerk-Übertragungskanälen kann ausserdem das SSL-Protokoll (Secure Socket Layer) benutzt werden, um die Sicherheit weiter zu erhöhen.

Ein Gültigkeitsende-Attribut ("exp") kann verwendet werden, welches den Zeitpunkt definiert, an dem oder nach dem ein Datagramm ungültig wird. Ein Gültigkeitsende kann besonders sinnvoll sein, um die Sicherheit bei der Manipulation von Aktuatoren zu erhöhen. Ohne Gültigkeitsende kann ein auf einem "angezapften" Übertragungskanal "mitgehörtes" Datagramm (selbst dann, wenn es mit einer digitalen Signatur gesichert ist) identisch zu einem späteren Zeitpunkt erneut gesendet werden, um eine bestimmte Aktion zu wiederholen (zum Beispiel ein Tor, eine Sicherheitsschleuse usw. zu öffnen). Die Angabe eines Gültigkeitsendes (zum Beispiel eine Sekunde nachdem das Datagramm abgesendet wurde) kann einen Schutz gegen missbräuchlich erneut übertragene Datagramme bieten, da sie nach dem Gültigkeitsende vom Empfänger nicht bearbeitet werden.

Einige Implementierungen der vorliegenden Erfindung können gespeicherte Profile verwenden. Profile können allgemein dazu benutzt werden, um die Gesamtheit der Elemente inklusive zulässiger Wertebereiche zu beschreiben, die in Datagrammen für ein bestimmtes Gerät oder eine bestimmte Anwendung benutzt werden können. FactoryXML-Profile können in unterschiedlicher Form spezifiziert werden. Zum Beispiel kann ein FactoryXML-Profil als XML Document Type Definition (DTD) ausgedrückt werden. Alternativ hierzu kann ein FactoryXML-Profil als XML Schema ausgedrückt werden. Bevorzugterweise modelliert ein Profil eine bestimmte Systemkomponente so vollständig wie möglich. In einigen Implementierungen, können generische Profile als Ausgangspunkt für individuelle Profile benutzt werden. In anderen Implementierungen können FactoryXML-Profile auf der Basis standardisierter Profile modelliert werden, die von Standardisierungsgremien für verschiedene Anwendungszwecke, Gerätetypen und Branchen definiert wurden.

Ein beispielhaftes FactoryXML-Profil kann wie folgt aussehen:

```
<!DOCTYPE sensor [
<!ELEMENT sensor (value, unit, error?)>
<!ATTLIST sensor src CDATA #IMPLIED dest CDATA
 #IMPLIED t CDATA #IMPLIED seq CDATA #IMPLIED
 crc CDATA #IMPLIED>
<!ELEMENT value (#CDATA )?>
<!ELEMENT unit ANY>
<!ELEMENT error (#CDATA)>
]>
```

Das dargestellte Profil ist ein generisches FactoryXML-Profil für Sensoren (wie zum Beispiel Temperaturfühler, Drucksensoren, Spannungsmesser, Waagen usw.), bestehend aus einem Element <value> und einem Element <unit>. Ein Beispieldatagramm für einen Temperatursensor sieht folgendermassen aus:

```
<sensor src="mixer2" t="181234.90Z" seq="42537">
        <value>45.7553</value>
        <unit><Fahrenheit/></unit?>
</sensor>
```

Als eine Implementierungsmöglichkeit kann ein generisches FactoryXML-Profil für Aktoren (wie Schalter, Ventile usw.) mit einem <value>-Element wie folgt implementiert werden:

```
<!DOCTYPE actuator [
<!ELEMENT actuator (value, error?)>
<!ATTLIST actuator src CDATA #IMPLIED t CDATA #IMPLIED
seq CDATA #IMPLIED>
<!ELEMENT value (#CDATA)>
<!ELEMENT error (#CDATA)>
]>
```

Ein beispielhaftes Datagramm zur Manipulation eines digitalen (binären) Ausgangs (Schalters) mit diesem Profil kann wie folgt aussehen:

```
<set dest="Switch5">
        <actuator>
                <value><On/></value>
        </actuator>
</set>
```

Komplexe Peripheriegeräte wie SPSen können meist nur schwer mit generischen Profilen modelliert werden. Dennoch lassen sich anwendungsspezifische Profile für komplexe Peripheriegeräte definieren. Ein beispielhaftes FactoryXML-Profil für das Siemens-Bilderkennungssystem SIMATIC VS710 sieht folgendermassen aus:

```
<!DOCTYPE vs710 [
<!ELEMENT vs710 (program | state | monitor | readconfig | writeconfig |
  observation | error) ?>
<!ATTLIST vs710 src CDATA #IMPLIED dest CDATA #IMPLIED t
  CDATA #IMPLIED seq CDATA #IMPLIED sig CDATA #IMPLIED>
<!ELEMENT program (load | reset | start | stop)>
<!ELEMENT load (#CDATA)>
<!ELEMENT reset EMPTY>
```

```
<!ELEMENT start EMPTY>
<!ELEMENT stop EMPTY>
<!ELEMENT state (#CDATA)>
<!ELEMENT monitor (start | stop)>
<!ELEMENT readconfig (#CDATA)>
<!ELEMENT writeconfig (#CDATA)>
<!ELEMENT observation (bitpattern | partnumber | string | float)>
<!ELEMENT bitpattern (#CDATA)>
<!ELEMENT partnumber (#CDATA)>
<!ELEMENT string (#CDATA)>
<!ELEMENT float (#CDATA)>
<!ELEMENT error (#CDATA)>
]>
```

In dieser Implementierung besteht das Peripheriegerät aus einer Kamera mit einem integrierten Prozessor zur Bilderkennung, um Prozesse zum Beispiel in Zusammenhang mit Fliessbändern zu überwachen. Die VS710 hat ein PROFIBUS-Interface und eine asynchrone serielle Schnittstelle. Die asynchrone serielle Schnittstelle verwendet das 3964R-Protokoll zur Fehlerkorrektur und ein proprietäres, gerätespezifisches Protokoll oberhalb von 3964R zum Kommando- und Datenaustausch.

Wie gezeigt können FactoryXML-Profile so gestaltet werden, dass sie alle Funktionen und Eigenarten eines Geräts oder einer Anwendung beinhalten. Zum Beispiel kann das FactoryXML-Element "program" benutzt werden, um das Bilderkennungsprogramm innerhalb des VS710 zu manipulieren. Es verwendet die Unterelemente "load", "reset", "start" und "stop". Das FactoryXML-Element "monitor" startet oder stoppt die Ausgabe von Bilderkennungsergebnissen. Das FactoryXML-Element "observation" enthält Ergebnisse, die vom VS710 spontan erzeugt werden, wenn die Beobachtungsfunktion "monitor" eingeschaltet wurde. Die FactoryXML-Elemente "readconfig" und "writeconfig" können dazu benutzt werden, um Konfigurationsdaten zu lesen oder zu schreiben.

Optional können FactoryXML-Datagramme in einen FactoryXML-"Envelope" eingeschlossen werden. Ein FactoryXML-Envelope ist kein FactoryXML-Element, da er mehrfache identische Unterelemente enthalten kann. FactoryXML-Envelopes können darüber hinaus keine FactoryXML-Standardattribute enthalten. Ein FactoryXML-Envelope kann in Situationen hilfreich sein, in denen ein XML-Parser keine mehrfachen Wurzelelemente unterstützt. Ein beispielhaftes FactoryXML-Datagramm in einem FactoryXML-Envelope kann wie folgt aussehen:

```
<FactoryXML version="1.1">
        <status>
        <sensor src="mixer2">
                <value>44.534</value>
                <unit><Kilogram/></units>
        </sensor>
        </status>
</FactoryXML>
```

UTV

Die vorliegende Erfindung sieht die Verwendung alternativer Datenformate vor. Eines dieser alternativen Datenformat wird als UTV bezeichnet, ein Akronym for "unstructured tag/value". Ein sehr einfaches UTV-Datagramm kann wie folgt aussehen:

```
src = Thermometer2
t = 234412
value = 1.92
```

unit = Celsius

Wie man sieht, enthält UTV ähnlichen Inhalt wie FactoryXML, aber die Daten weisen eine geringere Strukturierung auf. Bei UTV-Codierung kann ein Tag von einem optionalen Wert gefolgt werden. Tags und Werte können zum Beispiel durch ein Gleichheitszeichen voneinander abgetrennt werden. Tags ohne Werte entsprechen leeren FactoryXML-Elementen. Vordefinierte Tags (wie <Kg/>, <On/>, <Off/>) können in UTV wie Werte behandelt werden. Multiple Tags können durch Leerzeichen (Whitespace) voneinander abgetrennt werden. Als Leerzeichen können gelten (ohne Anspruch auf Vollständigkeit): [SP] (ASCII-Zeichen 32), "und"-Zeichen ("&"), Pluszeichen ("+"), Tabulatoren, Wagenrückläufe ([CR], ASCII-Zeichen 13) und Zeilenvorschübe ([LF], ASCII-Zeichen 10). Elemente und Attribute werden in UTV im allgemeinen syntaktisch gleich behandelt. Das folgende Beispieldatagramm enthält den selben Inhalt wie das obige Beispiel, verwenden aber andere Trennzeichen:

src = Themometer2&t = 234412&value = 1.92&unit = Celsius

Einige UTV-Implementierungen können übergeordnete Elemente enthalten, um mehr Information und Struktur zu übermitteln. Übergeordnete Elemente können als Präfixe codiert werden, wobei zur Abtrennung des Präfixes ein Punkt verwendet wird. Ein beispielhaftes Datagramm dieser Implementierung kann wie folgt strukturiert sein:

sensor.src = Thermometer2
sensor.t = 234412
sensor.value = 1.92
sensor.unit = Celsius

UTV kann in Situationen verwendet werden, in denen die Zielanwendung nicht in der Lage ist, FactoryXML zu verarbeiten oder in denen FactoryXML einfach unpraktisch wäre. UTV erfordert paketorientierte Übertragungskanäle, bei denen das Übertragungsprotokoll die Kennzeichnung des Nachrichtenendes (bzw. Datagrammendes) unterstützt. Sinnvolle Übertragungskanäle für UTV beinhalten (ohne Ausschliesslichkeit): HTTP Clients, die Gerätezustände an einen HTTP-Server mittels URI-Codierung übermitteln (da URI-Codierung FactoryXML nicht unterstützt, wohl aber UTV); HTTP Server, die passive Transaktionen mittels URI-Codierung akzeptieren; SMS-Gateways (bidirektional), da aktive Alarm-Transaktionen UTV verwenden können, um Alarmnachrichten an Handy-Displays (als UTV-Datagramm codiert) zu übertragen, auf welche ein Benutzer mit der Rückübertragung einer UTV-codierten SMS-Nachricht antworten kann, die er auf der Tastatur seines Handies eingetippt hat und die auf der anderen Seite zu Konfigurationsänderungen oder ähnlichem führt; WAP/WML Clients aufgrund der eingeschränkten Darstellungsmöglichkeiten bei WAP; Alarmierung und Fernkonfiguration per Email; und einfache HTML-Seiten für Browser, die XML nicht unterstützen.

Transaktionen

Kommandos können an eine Automatisierungskomponente mit dem Standardelement "set" gesendet werden. In einer bevorzugten Implementierung wird ein Kommando mit dem neuen bzw. aktuellen Gerätezustand quittiert. Wenn das "seq"-Attribut (Sequenznummer) verwendet wird, wird die Automatisierungskomponente bevorzugterweise denselben "seq"-Wert in ihrem Antwortdatagramm verwenden. Die folgende Ablauftabelle verdeutlicht die Inhalte eines Datagrammaustauschs, bei dem eine DS ein Kommando an eine AK (zum Beispiel einen binären Ausgang) sendet, den Zustand "On" zu setzen:

Datenverarbeitende Station	**Automatisierungskomponente**
`<set dest="Switch5"> <actuator seq="12"> <value><On/></value> </actuator> </set>`	
	`<status> <actuator src="Switch5" seq="12"> <value><On/></value> </actuator> </status>`

In der dargestellten Transaktion sendet die DS ein Kommando an eine AK mit der Netzadresse "Switch5". Das Kommando lautet, dass der Inhalt des Elements <value> auf den Wert "On" gesetzt werden soll. Die DS verwendet die Sequenznummer 12 zur Kennzeichnung dieser Transaktion. Die AK antwortet mit einem Datagramm, welches den neuen Wert enthält, und zeigt damit an, dass das Kommando erfolgreich ausgeführt wurde. Das Antwortdatagramm verwendet dieselbe Sequenznummer wie das Kommandodatagramm, so dass die Antwort eindeutig dem Kommando zugeordnet werden kann.

Im Fehlerfall enthält das Antwortdatagramm den tatsächlichen Wert des betreffenden Elements, zusammen mit einem <error>-Element, welches eine Fehlerbeschreibung enthält. Die folgende Ablauftabelle stellt den Austausch von Datagrammen dar, bei dem eine DS Kommandos an eine AK (zum Beispiel einen digitalen Ausgang) schickt, um einen – nicht unterstützten – Zustand "high" einzuschalten.

Datenverarbeitende Station	Automatisierungskomponente
<set dest="Switch5"> <actuator seq="12"> <value>high</value> </actuator> </set>	
	<status> <actuator src="Switch5" seq="12"> <value><Off/></value> <error>illegal value</error> </actuator> </status>

Wie zu sehen ist, schlug diese Transaktion fehl, da die DS versucht hat, den Wert "high" einzustellen, die AK jedoch nur die Werte "On" und "Off" unterstützt.

Das Akzeptieren von Kommandos durch eine Systemkomponente kann per Konfiguration auf Kommandos von bestimmten Stationen (identifiziert durch ihre Quelladresse) beschränkt werden. Zusammen mit der Netzwerkadresse (sofern vorhanden) und mit der digitalen Signatur lässt sich die Kommandoübergabe auf ausgewählte Stationen beschränken, wodurch Missbrauchsmöglichkeiten eingeschränkt werden.

Das Standardelement <poll> kann dazu verwendet werden, einen Geräte- oder Anwendungszustand abzufragen. Eine Abfrage-Transaktion ist ähnlich einer Kommando-Transaktion. Wenn <poll> als leeres Element verwendet wird, enthält das Antwortdatagramm eine vollständige Datenstruktur des abgefragten Geräts bzw. der abgefragten Anwendung. <poll> kann jedoch auch dazu benutzt werden, bestimmte Unterelemente gezielt abzufragen. Die folgende Beispiel-Transaktion veranschaulicht den Inhalt von Datagrammen, mit denen eine DS ("erp") den Zustand einer AK ("ramp") abfragt:

Datenverarbeitende Station	Automatisierungskomponente
<poll src="erp" dest="ramp"/>	
	<status dest="erp"> <sensor src="ramp" t="092312"> <value>545.32</value> <unit><Kg/></unit> </sensor> </status>

Die folgende Beispiel-Transaktion zeigt, wie eine DS den Inhalt eines bestimmten Unterelements abfragen kann:

Datenverarbeitende Station	Automatisierungskomponente
<poll src="erp" dest="ramp"> <sensor><unit/></sensor> </poll>	
	<status dest="erp"> <sensor src="ramp" t="092312"> <unit><Kg/></unit> </sensor> </status>

Aktive Transaktionen

Aktive Transaktionen werden spontan von einer AK initiiert. Die Ausgabe erfolgt zyklisch, bei Zustandsänderung oder beim Eintreffen bestimmter Bedingungen. In bevorzugten Implementierungen können aktive Transaktionen (speziell solche, die durch Zustandsänderungen ausgelöst werden) durch die Verwendung von Quittungen gegen Datenverlust geschützt werden.

Bei einer aktiven zyklischen Datenausgabe kann eine AK Datagramme zyklisch in vordefinierten Zeitintervallen (zum Beispiel jede Sekunde) absenden. In vielen Situationen wird allerdings die aktive Ausgabe bei Zustandsänderungen vorzuziehen sein, da sie die Netzwerklast reduziert und die Programmierung der DS vereinfacht.

Bei einer aktiven Datenausgabe bei Zustandsänderungen kann eine AK ein Datagramm im Fall eines Zustandswechsels absenden. Ausserdem kann der aktuelle Zustand der AK beim Verbindungsaufbau oder beim Einschalten des Geräts (für verbindungslose Übertragungskanäle) gesendet werden. Aktive Datenausgabe bei Zustandsänderungen ermöglicht eine erhebliche Reduktion der Komplexität von Softwareanwendungen, die die Daten auswerten, und trägt ausserdem dazu bei, die Netzwerklast zu reduzieren. Für verbindungslose oder ungesicherte Übertragungskanäle können Quittungen benutzt werden, um Datenverluste zu vermeiden, die sonst zu schwerwiegenden Störungen führen könnten.

Bei der aktiven Datenausgabe beim Eintreten bestimmter Bedingungen (Alarmierung) kann eine AK ein Datagramm abschicken, wenn eine oder mehrere Bedingungen erfüllt sind. Die Bedingungen können einfach oder komplex sein.

In bevorzugten Implementierungen können aktive Transaktionen von der DS quittiert werden, um zu verhindern, dass Datagramme nicht verloren gehen, speziell wenn nicht fehlerkorrigierende Übertragungskanäle wie UDP verwendet werden. Im Quittungsbetrieb wartet eine AK auf eine Quittung für jedes Datagramm, bevor das nächste Datagramm übertragen wird. Eine Quittung kann ein leeres Wurzelelement mit der Sequenznummer des empfangenen Datagramms sein. Alternativ können die Standardelemente <ack> und <nak> verwendet werden. Der Vorteil hiervon ist, dass damit auch negative Quittungen möglich sind, wodurch eine Transaktion bei einem gestörten Übertragungskanal beschleunigt wird (der Sender muss nicht erst bis zum Ablauf seines Timeouts warten). Die folgende Transaktion veranschaulicht den Quittungsbetrieb:

Datenverarbeitende Station	**Automatisierungskomponente**
	`<status ack="yes" ref="iuf2"> <sensor src="ramp5" seq="12"> <value>442.23</value> <unit><Kg/></unit> </sensor> </status>`
`<ack dest="ramp5" ref="iuf2"/>`	

Bei Verwendung fehlererkennender Übertragungskanäle wie UDP braucht nicht unbedingt ein Prüfsummenattribut verwendet zu werden. Prüfsummen werden vorzugsweise auf nicht fehlergesicherten Übertragungskanälen wie seriellen Asynchronschnittstellen verwendet.

Wenn die datenerzeugende Station innerhalb einer vordefinierten Zeit (Timeout) keine Quittung erhält, kann das letzte Datagramm erneut übertragen werden, bis eine Quittung empfangen wird oder eine vordefinierte maximale Zahl von Wiederholungen erreicht ist. Keine Folgedatagramme sollten übertragen werden, bevor eine gültige Quittung empfangen wurde.

Quittungen sind besonders sinnvoll in Verbindung mit der aktiven Datenausgabe bei Zustandswechsel. Die aktive Datenausgabe bei Zustandswechsel ist vorteilhaft, da sie die Programmierung vereinfacht und die Netzlast reduziert. Diese Übertragungsart kann jedoch zu fatalen Konsequenzen führen, wenn Datenverluste unbemerkt bleiben. Der Quittungsbetrieb kann benutzt werden, um solche Datenverluste bei nicht fehlerkorrigierenden Übertragungskanälen wie UDP zu verhindern.

Anmeldung und Abmeldung (subscribe/unsubscribe)

Die Zieladresse für aktive Transaktionen kann typischerweise auf zwei Arten festgelegt werden: (1) fest konfiguriert innerhalb eines Gerätes oder einer Anwendung, (2) dynamisch durch Anmeldung und Abmeldung. Eine dynamische Anmeldung und Abmeldung ist insbesondere sinnvoll für verbindungslose Übertragungskanäle wie UDP. Sie kann mit den Standardelementen "subscribe" und "unsubscribe" implementiert werden. Eine Station, die sich bei einem ADD-Gerät mit "subscribe" anmeldet, erhält Ausgaben von diesem Gerät je nach der vorkonligurierten Betriebsart (zum Beispiel aktive zyklische Datenausgabe), bis sie sich mit "unsubscribe" wieder abmeldet.

An- und Abmeldung können ADD-Kommandos sein. Eine An- und Abmeldung kann von der AK quittiert werden. Die folgenden Attribute können bei der Anmeldung verwendet werden:

<!ATTLIST subscribe
trigger (cyclicldeltalalarm) #IMPLIED
ack (yeslno) #IMPLIED>

Das Attribut "trigger" legt fest, ob die Ausgabe zyklisch (cyclic), bei Zustandsänderungen (delta) oder beim Eintreffen bestimmter Bedingungen (alarm) erfolgt. Das Attribut "ack" legt fest, ob Quittungen gesendet werden oder nicht.

Einige Implementierungen der vorliegenden Erfindung können An- und Abmeldetransaktionen verwenden. Eine beispielhafte An- und Abmeldetransaktion kann wie folgt aussehen:

Datenverarbeitende Station	Automatisierungskomponente
<subscribe> <sensor src="erp" dest="ramp2"/> </subscribe>	
	<ack src="ramp2" dest="erp"/> <status dest="erp"> <sensor src="ramp2" t="092312"> <value>123.2</value> <unit><Kg/></unit> </sensor> </status> <status dest="erp"> <sensor src="ramp2" t="092313"> <value>123.2</value> <unit><Kg/></unit> </sensor> </status> <status dest="erp"> <sensor src="ramp2" t="092314"> <value>123.9</value> <unit><Kg/></unit> </sensor> </status>
<unsubscribe src="erp" dest="ramp2"/>	
	<ack src="ramp2" dest="erp"/>

In diesem Beispiel meldet sich eine DS bei einer Waage ("ramp2") an, die daraufhin zyklisch (jede Sekunde) Datagramme an die DS sendet, bis sich die DS mit "unsubscribe" wieder abmeldet. An- und Abmeldung werden von der AK quittiert.

In einigen Fällen kann die DS eine Transaktionsbetriebsart wählen, der sich von der voreingestellten Betriebsart unterscheidet. Dies geschieht dadurch, dass Werte für die Attribute "trigger" und "ack" gesetzt werden. Die folgende Ablauftabelle stellt eine beispielhafte Transaktion dar, die veranschaulicht, wie eine Einheit sich für aktive Datenausgabe bei Zustandswechsel mit Bestätigungen anmelden kann:

Datenverarbeitende Station	Automatisierungskomponente
<subscribe src="erp" dest="ramp2" trigger="delta" ack="yes"/>	
	<ack src="ramp2" dest="erp"/>
	<status dest="erp" ack="yes" ref="8if1"> <sensor src="ramp2" t="092312" seq="0"> <value>123.2</value> <unit><Kg/></unit> </sensor> </status>
<ack src="erp" dest="ramp2" ref="8if1"/>	
	<status dest="erp" ack="yes" ref="98p"> <sensor src="ramp2" t="092314" seq="1"> <value>123.9</value> <unit><Kg/></unit> </sensor> </status>
<ack src="erp" dest="ramp2" ref="98p"/>	
<unsubscribe src="erp" dest="ramp2"/>	
	<ack src="ramp2" dest="erp" t="092314"/>

Hier meldet sich die DS für aktive Datenausgabe bei Zustandswechsel mit Bestätigungen an. Die AK beginnt die Ausgabe mit der Sequenznummer 0. Das Datagramm wird von der DS quittiert. Um 09:23:13 Uhr gibt es keine Übertragung, da sich der Gerätezustand (in diesem Fall das gemessene Gewicht) nicht geändert hat. Die nächste Übertragung erfolgt um 09:23:14 Uhr, da sich das Gewicht von 123.2 auf 123.9 Kilogramm verändert hat. Wieder wird das Datagramm quittiert, und anschliessend meldet sich die DS ab.

Bei der Anmeldung kann auch ein gültiger Wertebereich für ein bestimmtes Element festgelegt werden. Dies ist sinnvoll für Alarmierung, wobei unterschiedliche Anwendungen unterschiedliche Schwellwerte definieren können. Im Alarmierungsbetrieb wird eine Ausgabe ausgelöst, wenn Werte ausserhalb des zulässigen Bereichs fallen. Die Ausgabe wird bei Zustandswechseln fortgesetzt, bis der Wert des betreffenden Elements wieder innerhalb des gültigen Bereichs liegt, oder die Ausgabe kann zyklisch für eine vordefinierte Zeit fortgesetzt werden. Zur Definition eines gültigen Wertebereichs können die Standardelemente "min", "max", "equal" und "unequal" benutzt werden.

Die folgenden Beispiele erläutern (sowohl in FactoryXML als auch in UTV) Datagramme zur Anmeldung, die in unterschiedlichen Situationen benutzt werden können. Im folgenden Beispiel meldet sich eine DS für Alarmierung an, falls der Wert des Elements "temperature" den Wert 110.9 überschreitet oder den Wert 34.2 unterschreitet:

FactoryXML	UTV
<subscribe trigger="alarm" ack="no"> <temperature> <min>34.2</min> <max>110.9</max> </temperature> </subscribe>	subscribe.trigger=alarm subscribe.ack=no subscribe.temperature.min=34.2 subscribe.temperature.max=110.9

Das folgende Beispiel zeigt, wie sich eine Anwendung für Alarmierung anmelden kann, wenn der Wert des Elements "gate5" auf den Wert "Open" wechselt:

FactoryXML	UTV
<subscribe trigger="alarm" ack="no> <gate5> <equal><Open/></equal> </gate5> </subscribe>	subscribe.trigger=alarm subscribe.ack=no subscribe.gate5.equal=Open

Das nächste Beispiel zeigt, wie sich eine Anwendung für Alarmierung anmelden kann, wenn das Element "color" einen anderen Wert als "green" einnimmt:

FactoryXML	UTV
<subscribe trigger="alarm" ack="no> <color> <unequal>green</unequal></color> </subscribe>	subscribe.trigger=alarm subscribe.ack=no subscribe.color.unequal=green

Die Definition von Alarmbedingungen kann die Elemente "max", "min", "equal" und "unequal" verwenden. Das Vermischen dieser Elemente (mit Ausnahme von "max" und "min") innerhalb einer Bedingungsdefinition ist unzulässig und unsinnig. "max" und "min" können zusammen oder einzeln benutzt werden.

Übertragungskanäle

Keines der Datenformate, die in der vorliegenden Patentschrift beschrieben sind, ist an ein bestimmtes Transportmedium oder -protokoll gebunden. Dennoch werden in der Praxis bevorzugt Implementierungen mit TCP/IP oder verwandten Internet-Protokollen anzutreffen sein.

Implementierungen der vorliegenden Erfindung können mehrere unterschiedliche Übertragungskanäle verwenden, einschliesslich einfacher Transportprotokolle (ISO/OSI-Schicht 4) und Anwendungsprotokolle (ISO/OSI-Schicht 7). In der Praxis werden bevorzugt Implementierungen mit Transportprotokollen anzutreffen sein, da sie flexiblere und effizientere Möglichkeiten des Datenaustauschs erlauben.

In einer Form der Implementierung fungiert die AK als TCP Server und wartet auf Verbindungsanfragen. Eine DS kann eine Verbindung als TCP Client aufbauen, um Kommandos an die AK zu senden und/oder um Daten von der AK zu empfangen. Diese Art des Übertragungskanals kann alle FactoryXML-Transaktionen unterstützen. Ein Quittungsbetrieb

ist hier allerdings nicht sinnvoll, da TCP bereits fehlerkorrigierend ist. Über diesen Kanal kann auch eine An- und Abmeldung (subscribe/unsubscribe) erfolgen, wobei die AK aktiv eine TCP-Client-Verbindung zur DS aufbaut, ein Datagramm übermittelt, und die TCP-Verbindung wieder schliesst.

In einer anderen Implementierung kann die AK als TCP Client fungieren und Verbindungen zu einem vorkonfigurierten TCP Server aufbauen, um Datagramme zu übermitteln. Als Alternative zu fest konfigurierten Verbindungen können auch Verbindungen zu DS aufgebaut werden, die sich zuvor über einen TCP-Serverkanal bei der AK angemeldet haben (subscribe). Dieser Übertragungskanal ist in erster Linie sinnvoll für aktive (spontane) Transaktionen. Die Verwendung von Bestätigungsdatagrammen ist auf diesem Übertragungskanal nicht sinnvoll, da TCP bereits fehlerkorrigierend ist.

Anstelle von TCP kann auch UDP verwendet werden. UDP hat gegenüber TCP einige Vorteile, unter anderem bessere Performance durch geringeren Protokoll-Overhead, paketorientierte gegenüber datenstromorientierter Übertragung und Unterstützung von Broadcasts. Auf der anderen Seite ist UDP zwar fehlererkennend, aber nicht fehlerkorrigierend, so dass Paketverluste als Folge von Übertragungsfehlern auftreten können. Ausserdem können UDP-Datagramme beim Empfänger in der falschen Reihenfolge eintreffen. FactoryXML löst diese Probleme durch die Verwendung von Sequenznummern. Sequenznummern können dazu verwendet werden, Datenverluste und Sequenzfehler zu erkennen.

Datenverluste sind im Abfragebetrieb ("poll") möglicherweise nicht kritisch, da die DS das Ausbleiben einer Antwort auf ihre Abfrage erkennt und die Abfrage darauf hin erneut starten kann. Auch bei der aktiven (spontanen) zyklischen Ausgabe von Daten ist der Verlust einzelner Datenpakete normalerweise unkritisch, da ein "Update" der Daten automatisch im nächsten Zyklus erfolgt. Werden bei der spontanen Datenausgabe nur Änderungsdaten (oder Alarmierungsdaten) übertragen, empfiehlt sich dagegen die Verwendung eines anderen Transportprotokolls als UDP oder die Benutzung des Quittungsbetriebs. In dieser Betriebsart ist der mit den Quittungen verbundene Protokolloverhead für gewöhnlich akzeptabel, da eine Änderung der Daten, die zum Absenden eines Datagramms führt, vergleichsweise selten in den Betriebsarten auftritt, in denen dieses Verfahren normalerweise benutzt wird.

In Verbindung mit UDP sind die Anmeldung und Abmeldung für spontane Datagramme (subscribe/unsubscribe) besonders sinnvoll. Eine DS kann sich als UDP Client bei einer AK, die als UDP Server fungiert, anmelden (subscribe). Nach der Anmeldung werden die Rollen vertauscht und die AK sendet Änderungsdaten an die DS, die dann UDP-Datagramme empfängt.

Alternativ zu TCP und UDP können auch andere Internet-Protokolle wie zum Beispiel T/TCP (TCP for Transactions, siehe RFC 1379 und RFC 1644) verwendet werden. T/TCP bietet einen sehr viel schnelleren Verbindungsauf- und -abbau als TCP und ist von der Performance her mit UDP vergleichbar, ist jedoch fehlerkorrigierend wie TCP. Im Hinblick auf die vorliegende Erfindung ist T/TCP eine bevorzugte schnellere Alternative zu TCP.

Sowohl FactoryXML als auch UTV können beliebige datenstromorientierte Transportkanäle nutzen, wie zum Beispiel serielle Asynchronschnittstellen, ISDN oder USB. In der Praxis werden Implementierungen mit fehlerkorrigierenden Protokollen bevorzugt. Bei der Verwendung von fehlererkennenden, aber nicht fehlerkorrigierenden Protokollen können die in FactoryXML vorgesehenen Quittungen verwendet werden, um Datenverluste zu vermeiden. Für UTV sind fehlerkorrigierende, paketorientierte Übertragungskanäle erforderlich.

Implementienmgen der vorliegenden Erfindung können auch HTTP als Übertragungskanal nutzen. Der Vorteil von HTTP ist die grosse Anzahl von installierten HTTP Servern und Clients (d. h. Web-Browsern) weltweit. Der folgende Abschnitt beschreibt unterschiedliche Arten der Benutzung von HTTP zusammen mit der vorliegenden Erfindung, stellt dabei jedoch keine vollständige Liste möglicher Impementierungen dar.

HTTP Server, POST-Methode

Ein HTTP-POST-Befehl einer DS (der HTTP Client) kann benutzt werden, um ein FactoryXML-Datagramm (zum Beispiel mit einem <set>- oder <poll>-Datagramm) zu einer AK (dem HTTP Server) zu schicken. Die AK kann mit einem FactoryXML-Datagramm antworten. Eine weitergehende Datenausgabe der AK ist nur möglich, wenn die unterliegende TCP-Verbindung offen gehalten wird.

HTTP Server, GET-Methode

Eine HTTP-GET-Anfrage (zum Beispiel mit einem "set"- oder "poll"-Standardelement) kann von einer DS ausgelöst werden, wobei ein UTV-Datagramm innerhalb des URI transportiert wird. Beispiele hierfür sind die URIs "http://lightswitch5?set.actuator.value = on"und "http://themometer2?poll.sensor.unit". Die AK, die als HTTP Server fungiert, kann mit einem FactoryXML-Datagramm antworten. Weitere Ausgaben der AK sind möglich, sofern die unterliegende TCP-Verbindung geöffnet bleibt.

HTTP Client, POST-Methode

Dieser Übertragungskanal ist sinnvoll für Anwendungen, bei denen eine AK Daten durch Ausfüllen eines "Web-Formulars" übermitteln soll, oder wo einfach die weit verbreitete CGI-Technik genutzt werden soll, um betriebswirtschaftlichen Anwendungen Automatisierungsdaten zur Verfügung zu stellen. Der HTTP Server kann zum Beispiel eine betriebswirtschaftliche Anwendung sein, die so konfiguriert ist, dass sie die Sendedaten einer AK (zum Beispiel ein UTV-Datagramm) mit einem CGI-Script verarbeiten kann. Die AK (der HTTP Client) muss in geeigneter Form für aktive Datenausgabe konfiguriert sein, in der Regel für Ausgaben bei Zustandswechsel oder beim Eintreten vordefinierter Bedingungen.

HTTP Client, GET-Methode

Eine AK meldet sich als HTTP Client bei einer DS (dem HTTP Server) an und übermittelt ein UTV-Datagramm als

Teil des URI. Ein Beispiel hierfür wäre der URI "http://server?src = thermometer2&value = 23.442&unit = Celsius". Der HTTP Server kann darauf mit einem FactoryXML-Datagramm antworten, welches von einem CGI-Script generiert wird. Dieser Übertragungskanal ist sinnvoll für aktive Datenausgabe bei Zustandswechsel oder beim Eintreten vordefinierter Kriterien.

Email kann als Übertragungskanal benutzt werden und ist besonders sinnvoll für Alarmierung und Fernwartung. Email-Versand kann für aktive Datenausgabe einer AK genutzt werden. Die AK kann ein UTV-Datagramm zu einer vorkonfigurierten Email-Adresse senden oder an einen entfernten Benutzer, der sich per "subscribe" angemeldet hat. Email-Empfang kann für Fernkonfiguration verwendet werden. Ein entfernter Administrator kann Befehle in UTV- (oder FactoryXML-) Codierung absenden, die mit dem Internet-Email-Protokoll übertragen werden können. Email-Empfang kann ausserdem für die Anmeldung ("subscribe") für aktive Datenausgabe verwendet werden, wobei die aktive Datenausgabe dann an die Email-Adresse des entfernten Administrators per Email-Versand geschickt werden kann. Kommandos könne sowohl in das Betreff-Feld der Email als auch in den Email-Inhalt geschrieben werden.

Andere Implementierungen der vorliegenden Erfindung verwenden drahtlose Übertragungskanäle wie SMS und WAP. Handys haben eine besondere praktische Bedeutung für Alarmierung, Fernwartung und Fernkonfiguration. Praktisch jeder Wartungsingenieur im Aussendienst verfügt über ein Handy. Ausserdem sind Handies in der Lage, ASCII-Textnachrichten (SMS) zu empfangen und zu senden und dienen als eingeschränkte Zugangsmöglichkeit zum Internet per WAP. Beide Möglichkeiten machen sie zu geeigneten und attraktiven Terminals für ADD.

SMS-Nachrichten sind kurze Textnachrichten (maximal 160 Zeichen) die auf dem Handy-Display ausgegeben werden können oder auf der Tastatur des Handies eingegeben werden können. Die aktive Datenausgabe einer AK aufgrund von Zustandswechseln oder aufgrund des Eintretens vordefinierter Bedingungen kann UTV-codiert zu einem oder mehreren Handys per SMS geschickt werden. Je nach SMS Service Center kann es unterschiedliche Zugangsmöglichkeiten geben, um in das Funknetzwerk zu gelangen. Die Zugangsmöglichkeiten umfassen normalerweise Email, Analogmodem, ISDN, und GSM-Modem; alle diese Möglichkeiten können auf einfache Weise von ADD-Geräten unterstützt werden. Für alle Zugangsmöglichkeiten zum SMS Service Center ausser Email kann es wünschenswert sein, ein SMS-Netzwerk-Gateway zu installieren, welches mehrere Anwendungen bzw. Geräte innerhalb des Netzwerks benutzen können. AK können per Handy durch Verwendung von SMS-Nachrichten manipuliert werden.

Einige Handies können bereits im Internet "Browsen", indem sie das Wireless Application Protocol (WAP) für Datenübertragung unterstützen und die Wireless Markup Language (WML) zur Darstellung der Daten. WML ist eine Untermenge von HTML. WAP ermöglicht die Abfrage von ADD-Geräten durch "Browsen" durch Menüs und Hotlinks und kann für Ferndiagnose und Fernwartung eingesetzt werden.

WAP ist jedoch ungeeignet für Alarmierung und aktive Datenausgabe in einigen Implementierungen, da ein Peripheriegerät per WAP möglicherweise nicht aktiv an ein Handy senden kann.

Die vorstehenden Beispiele wurden geschildert, um bestimmte Implementierungen der Erfindung zu erläutern. Technisch kundige Leser sollten die in den Beispielen dargestellten Techniken bitte als solche verstehen, die der Erfinder als gut funktionierend für die praktische Umsetzung der Erfindung erkannt hat und die deshalb als bevorzugte Verfahrensweisen bei ihrer praktischen Umsetzung betrachtet werden können. Technisch kundige Leser sollten jedoch auch verstehen, dass zahlreiche Änderungen innerhalb bestimmter hier dargelegter Implementierungen der Erfindung vorgenommen werden können, die dennoch zu einem gleichen oder ähnlichen Resultat führen, ohne vom Geist und vom Geltungsbereich der Erfindung abzuweichen.

Patentansprüche

1. Ein System zur Verteilung von Automatisierungsdaten (ADD), bestehend aus:
einer datenverarbeitenden Station (DS) mit einer Netzwerkadresse;
einer Automatisierungskomponente (AK) mit einer Netzwerkadresse und funktional verbunden mit der DS;
in dem besagte DS und besagte AK so konfiguriert sind, dass sie mittels einer oder mehrerer Transaktionen miteinander kommunizieren, wobei besagte Transaktion aus einem Datagramm besteht, wobei das Datagramm vollständig codiert ist und eine Beschreibung in Klarschrift enthält.

2. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren ein Quell-Attribut zur Kennzeichnung der Netzwerkadresse der Quelle des Datagramms enthält.

3. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren ein Ziel-Attribut enthält, welches die Netzwerkadresse der intendierten Datensenke für dieses Datagramm enthält.

4. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren enthält:
ein Wert-Attribut, sowie
ein Einheit-Attribut, welches die von besagtem Wert-Attribut verwendete Einheit angibt.

5. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren ein oder mehrere Sichheitsattribute enthält.

6. Das ADD-System nach Anspruch 5. in dem mindestens eines der besagten Sicherheitsattribute aus einem Attribut mit einer digitalen Signatur besteht.

7. Das ADD-System nach Anspruch 5. in dem mindestens eines der besagten Sicherheitsattribute aus einem Gültigkeitsende-Attribut besteht, welches einen Zeitpunkt angibt, nach welchem besagtes Datagramm ungültig wird.

8. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren ein Zeitstempel-Attribut beinhaltet, welches den Zeitpunkt angibt, an dem besagtes Datagramm erzeugt wurde.

9. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm des weiteren ein oder mehrere Attribute zur Fehlersicherung beinhaltet.

10. Das ADD-System nach Anspruch 9, in dem besagtes Datagramm des weiteren ein Prüfsummen-Attribut umfasst, welches Redundanzinformation des Inhalts von besagtem Datagramm enthält.

11. Das ADD-System nach Anspruch 9, in dem besagtes Datagramm des weiteren ein Sequenznummer-Attribut

enthält.

12. Das ADD-System nach Anspruch 1, welches des weiteren ein Profil enthält, welches die in besagtem Datagramm zulässigen Elemente, Parameter und Attribute spezifiziert.

13. Das ADD-System nach Anspruch 12, in welchem besagtes Profil ein Datagramm zur Verwendung mit einer bestimmten Art AK spezifiziert.

14. Das ADD-System nach Anspruch 1, in welchem besagtes Datagramm einen Zustand besagter AK repräsentiert.

15. Das ADD-System nach Anspruch 1, in welchem besagtes Datagramm ein Kommando von besagter DS an besagte AK repräsentiert.

16. Das ADD-System nach Anspruch 1, in dem besagte Transaktion des weiteren ein Quittungsdatagramm beinhaltet, mit dem der erfolgreiche oder nicht erfolgreiche Empfang besagten Datagramms angezeigt werden kann.

17. Das ADD-System nach Anspruch 1, in dem besagte Transaktion(en) eine aktive Transaktion ist.

18. Das ADD-System nach Anspruch 1, in dem besagte Transaktion(en) eine passive Transaktion ist.

19. Das ADD-System nach Anspruch 1, welches des weiteren eine Koordinierungsstelle enthält, welche funktional mit besagter/besagten DS und mit besagter/besagten AK gekoppelt ist, wobei besagte Koordinierungsstelle besagten DS und AK Netzwerkadressen zuweisen kann.

20. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm das FactoryXML-Format verwendet.

21. Das ADD-System nach Anspruch 1, in dem besagtes Datagramm das UTV-Format verwendet.

22. Ein System zur Verteilung von Automatisierungsdaten, bestehend aus:
eine Planungsebene, bestehend aus
einem Planungsnetzwerk, und einer oder mehrerer DS, die funktional mit besagtem Planungsnetzwerk gekoppelt sind;
einer Steuerungsebene, bestehend aus
einem Steuerungsnetzwerk und einer oder mehreren AK, die funktional mit besagtem Planungsnetzwerk gekoppelt sind; und
einer oder mehrerer Gateway-Einheiten, die funktional mit besagtem/n AK
und mit besagtem Planungsnetzwerk verknüpft sind,
wobei besagte Gateway-Einheit(en) dazu dienen, besagte DS darin zu unterstützen, mit besagten AK zu kommunizieren, indem sie ein oder mehrere Datagramme generieren und zu besagten AK senden, wobei die Datagramme vollständig codierte Beschreibungen in Klarschrift enthalten; und
wobei besagte AK mit besagter/n DS dadurch kommunizieren können, dass sie ein oder mehrere Datagramme durch besagte Gateway-Einheiten zu besagten DS senden können.

23. Ein Datagramm zur Verwendung in einem System zur Verteilung von Automatisierungsdaten, bestehend aus einem AK, welches funktional mit einer DS gekoppelt ist, wobei das Datagramm beinhaltet:
ein Quell-Attribut, welches die Datenquelle besagten Datagramms identifiziert, wobei besagte Datenquelle die AK oder die DS sein kann;
ein Ziel-Attribut, welches eine Datensenke besagten Datagramms identifiziert, wobei besagte Datensenke die AK oder die DS sein kann;
wobei besagtes Datagramm vollständig codiert ist.

24. Ein Verfahren zur automatisierten Datenverteilung in einem System bestehend aus einer Quelleinheit, welche funktional mit einer Zieleinheit gekoppelt ist, wobei das Verfahren beinhaltet:
eine Quelleinheit, die ein Datagramm generiert, welches aus vollständig codierten Beschreibungen in Klarschrift besteht; und
einer Zieleinheit, die besagtes Datagramm empfängt und auf den Inhalt besagten Datagramms sinnvoll antwortet.

25. Das Verfahren zur automatisierten Datenübertragung nach Anspruch 24, in dem besagte Quelleinheit eine DS ist, welche ein Abfrage-Datagramm an besagte Zieleinheit (eine AK) sendet, und besagte Zieleinheit mit dem Senden eines Datagramms zu besagter Quelleinheit antwortet, wobei das Datagramm ein Wert-Attribut und ein zugehöriges Einheiten-Attribut enthält.

26. Das Verfahren zur automatisierten Datenverteilung nach Anspruch 24, in dem des weiteren besagte Quelleinheit eine AK ist, die aktiv ein oder mehrere Datagramme generiert und besagte ein oder mehrere Datagramme an besagte Zieleinheit beim Eintreten eines vordefinierten auslösenden Ereignisses sendet.

Hierzu 3 Seite(n) Zeichnungen

Nummer: DE 101 12 843 A1
Int. Cl.[7]: G 06 F 17/60
Offenlegungstag: 27. September 2001

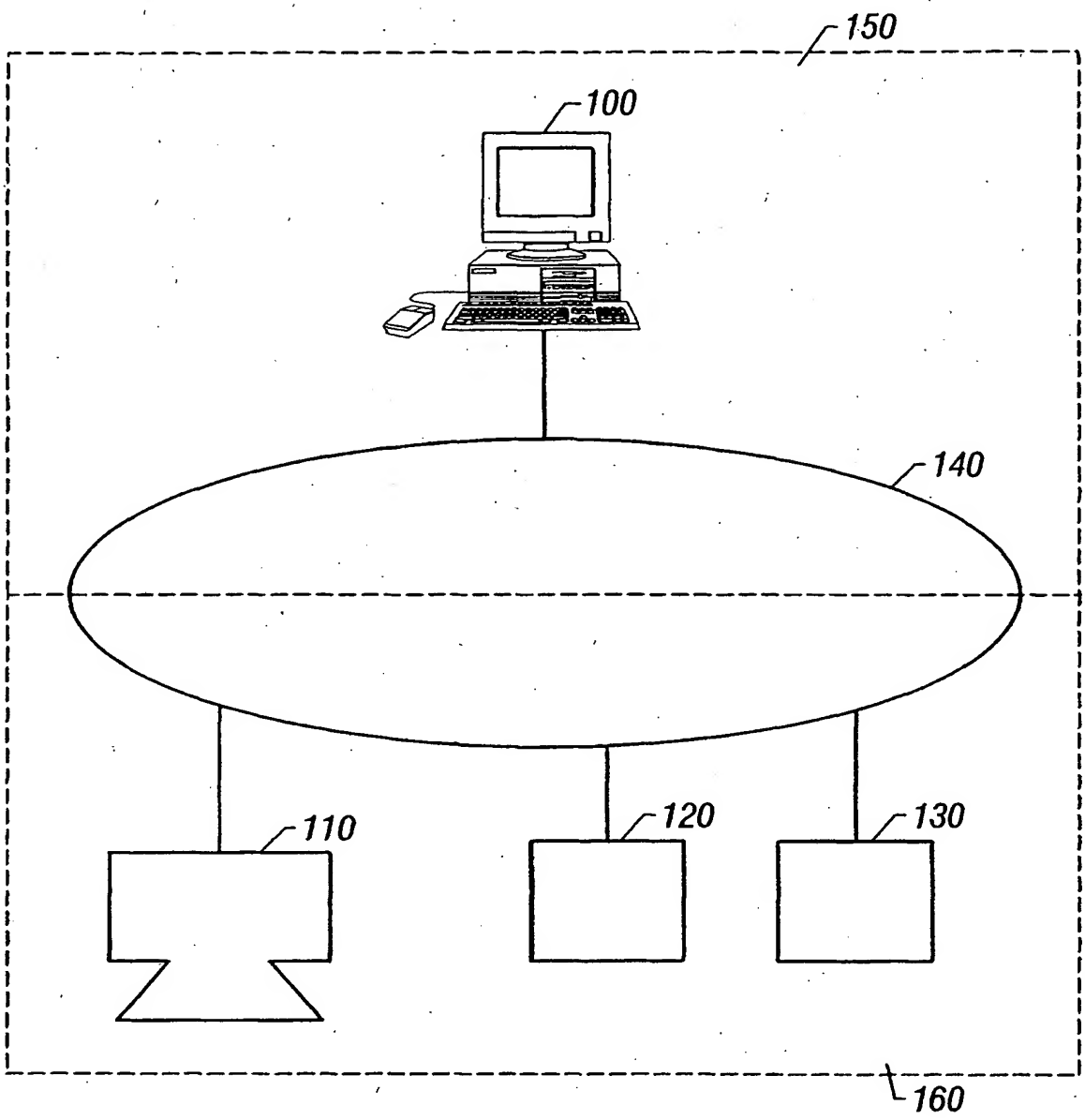


FIG. 1

Nummer: DE 101 12 843 A1
Int. Cl.[7]: G 06 F 17/60
Offenlegungstag: 27. September 2001

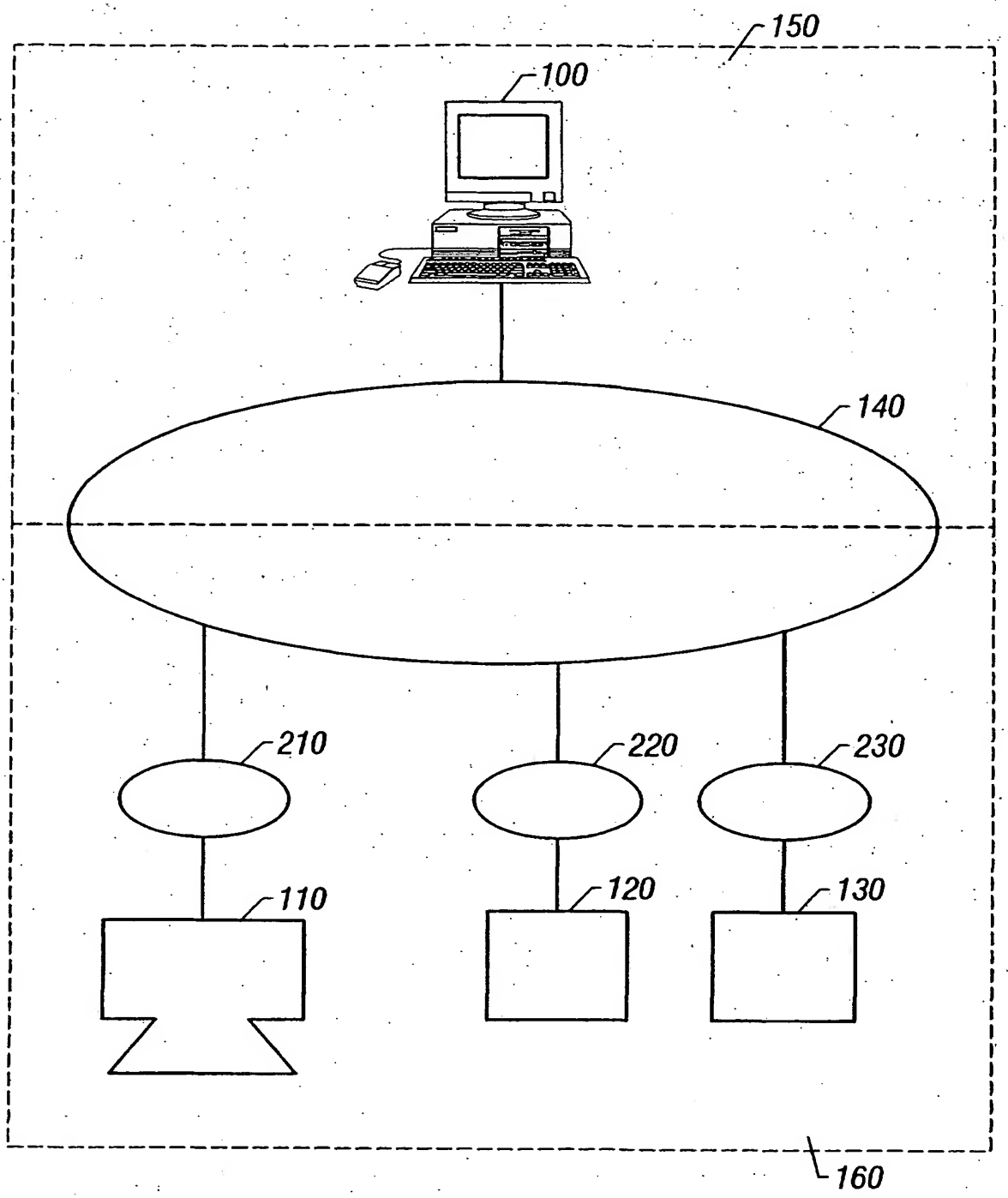


FIG. 2

Nummer: DE 101 12 843 A1
Int. Cl.[7]: G 06 F 17/60
Offenlegungstag: 27. September 2001

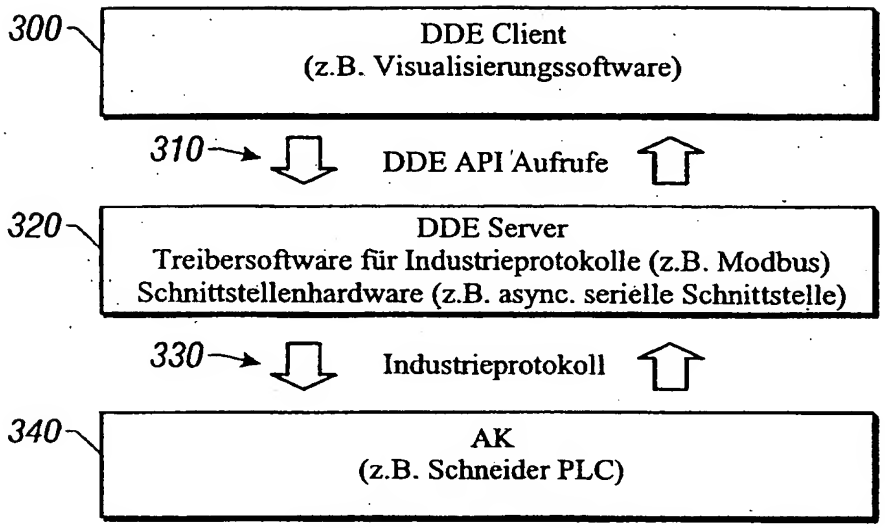


FIG. 3

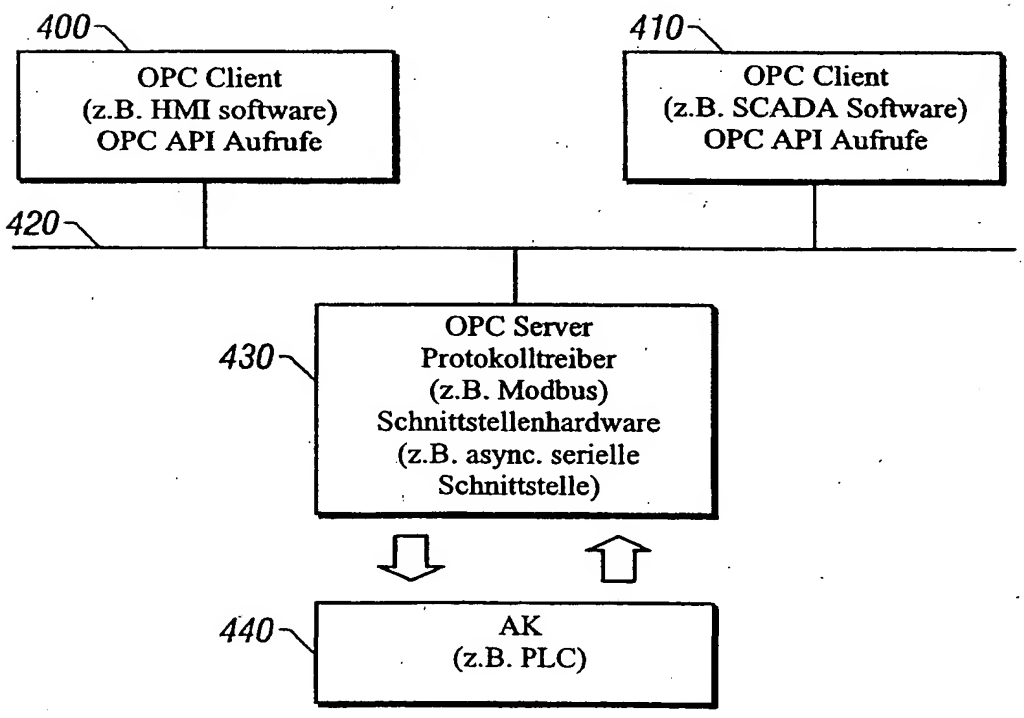


FIG. 4